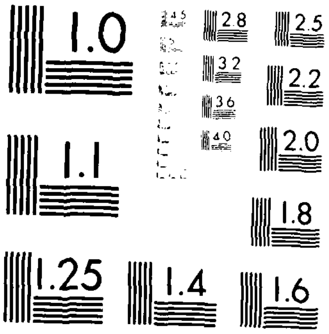
AD-A151 961 SYSTEM DESIGN OF AUTOMATED VLSI (VERY LARGE SCALE 1/4
INTEGRATED) TEST STATIO..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. S TARIQ
UNCLASSIFIED DEC 84 AFIT/GE/EE/84D-27 F/G 9/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

SYSTEM DESIGN OF AUTOMATED VLSI TEST STATION AND IMPLEMENTATION OF SELECTED SYSTEM ASPECTS

THESIS

AFIT/GE/EE/84D-27 Saleem Tariq
Sqn.Ldr. PAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/84D-27

SYSTEM DESIGN OF AUTOMATED VLSI TEST STATION AND IMPLEMENTATION OF SELECTED SYSTEM ASPECTS

THESIS

AFIT/GE/EE/84D-27 Saleem Tariq
Sqn.Ldr. PAF

Approved for release; distribution unlimited.

AFIT/GE/EE/84D-27

SYSTEM DESIGN OF AUTOMATED VLSI TEST STATION AND IMPLEMENTATION OF SELECTED SYSTEM ASPECTS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Saleem Tariq, B.S.
Sqn. Ldr. PAF
Graduate Electrical Engineering

December 1984

Approved for public release; distribution unlimited.

Preface

This report is system design of a capability to test VLSI circuits and implementation of two selected aspects of the system design. I undertook the project because it provided an opportunity for me to realize the major steps and amount of work involved in setting up a VLSI test station. Originally, I started this project with only its hardware aspect in view. As the project developed, whole effort was directed towards system design and program development. This change of direction provided for me my first experience to design a project with system outlook.

The system design has been partially implemented and it is hoped that its implementation will be completed in coming years to fit this Automated Test Station for VLSI (ATV) in long term scheme of Design Automation curricula at AFIT.

It is a pleasure to acknowledge my indebtedness to my faculty advisor, Lt. Col. Hal Carter, who often gave me a free hand to achieve the objectives. He provided excellent guidance at the beginning of this project while establishing the system requirements and was always willing to provide help as the project developed.

Finally, I want to express my gratitude to my wife, Beena, and daughter Mariam for their patience and understanding.

Tariq Saleem

Table of Contents

List of Figures

Figure Page

List of Tables

Abstract

Automated Test Station for VLSI (ATV) is a system design to ascertain correct functioning of a VLSI circuit. It is intended to test an Integrated Circuit (VLSI) by using Stanford IC Tester, (developed at Stanford University, California). The tester has the capability of addressing, simulating, and measuring status of any pin of its test connector, to which an ICUT (IC Under Test) is attached.

The test vectors to simulate the ICUT and reference data to analyze the response of an ICUT are extracted from ESIM files in VAX-11/780 computer system and stored on 8" - floppy disks to be utilized with microcomputer. These ESIM files, typically produced during Computer Aided Design phase of a VLSI circuit, contain node data generated during its simulator run.

The LSI-11/23 microcomputer will be used to control the functions of IC tester and provide test and reference data. The user will have the capability to guide the course of operation by selecting various operating options in an interactive manner.

SYSTEM DESIGN OF AUTOMATED VLSI TEST STATION AND IMPLEMENTATION OF SELECTED SYSTEM ASPECTS

I. Introduction

This project is the design of a capability to test VLSI circuits at AFIT. The requirement of ascertaining the correct functioning of a VLSI circuit is on the rise in step with the increasing complexity of these circuits. An integrated circuit is functionally tested by simulating it with a known input sequence and then comparing its output with a reference data sequence.

A VLSI Chip may consist of more than 100,000 components organized in a number of complex functional stages between its input and output pins. Additionally, the number of input and output pins may exceed a nominal value of 20, but it entails simulation of VLSI with potentially over 8 million test sequences.

This makes the manual mode of testing a VLSI circuit far too laborious, monotonous, and time consuming even to be considered. Problems like VLSI simulation are well suited for computer application; however, a discrete judgment has yet to be made to limit the number of test sequences necessary to ascertain the functioning of a circuit because the number of test sequences increase experimentally with the increase in the number of input pins. The simulation of a VLSI circuit with 20 input pins, sampling its outputs and comparing it with reference data, even if all three steps are completed within incredible limit of one microsecond, could require more then 150 days to exhaustively test one inch VSLI circuit.

Exhaustive testing requires not only unnecessary but most part of it unproductive efforts which can be eliminated by selecting a specific set of input test sequences.

This specific set of test sequences will be obtained from ESIM data files. These files are typically produced during the layout phase of the VLSI circuits and consist of input test data and resulting output data. These files are produced and available on the VAX 11/780 computer system (AFIT).

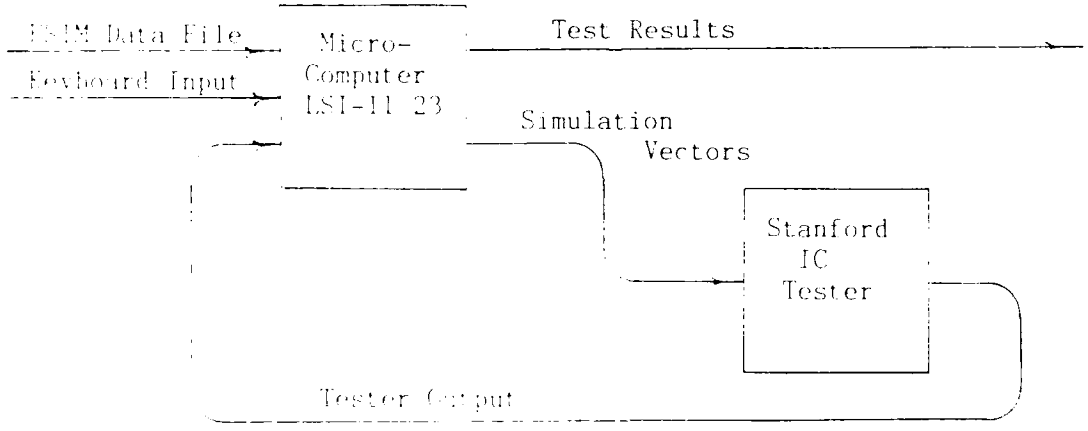


Figure I-1: Automated Tester for VLSI (ATV)

The remaining part of this chapter consists of the system node list. Node list consists of an indented index of the node numbers of all activity diagrams along with their corresponding titles. Activity diagrams represent the chosen set of processes in a system. The indentation serves as an outline of the system hierarchy. [Ref. 16]

The Node diagrams and their descriptions are attached as Appendix "A". The Data dictionary (Process Definitions and Data Flow Definitions) has been attached as Appendix "G". The Node List contains the names of all nodes whereas Node Descriptions describe function of each node in terms of its sub-modules. Process Definitions explain each node in terms of its input and output. Data Flow Definitions provide all information regarding data, which is passed from one node to another.

<u>System Node List</u>

- Node A-0: ATV - Automated Tester for VLSI circuits
- Node A0: ATV
    - Node A1: Extract Test Data
        - Node A11: Read ESIM File
            - Node A111: Get Text Line
            - Node A112: Classify Text Line
        - Node A12: Tabulate Command Data
            - Node A121: Check for Specific Command
            - Node A122: Tabulate Test Pins
            - Node A123: Handle Remaining Commands

points to represent binary sequences. The system will sample the output pins of ICUT after it has been simulated by a test vector and compare these output values with reference data - which is available from ESIM-file in "Auto" mode of operation only. The system will translate this "comparison" into appropriate test-messages. The sub-system (b) will operate independent of sub-system (a) and will be completely implemented on microcomputer LSI-11/23.

Both Sub-Systems were designed using structured Analysis Diagram Technique (SADTs). SADT is a comprehensive methodology for performing functional analysis and system design. It comprises of a number of coherent, integrated set of methods and rules that constitute a disciplined approach to analysis and design. [Ref. 16]

Technical details regarding symbolic representation and notation of SADTs are well documented in Ref. 3 and Ref. 16.

The structured design methodology was selected because it enables the whole system to be sub-divided into modules with definite and well-defined interfaces [Ref. 1, 8]. This scheme of design allows defining each module and its sub-modules in terms of input and output parameters.

The VLSI tester is composed of two major functional groups as shown in Figure II-2. Each group is further expanded, in stages, into its sub-modules to provide more insight of its design. Each successive level gives more information regarding how an input is being transformed into an output in this module.

file, will scan the given file and will classify the IC pins (test nodes) into three different categories of input, clock and output pins. The sub-system will dedicate memory buffers and store test data for each node into its respective buffer. The sub-system will finally change the available node data into test vector format, and write out these vectors into an external file. The "test vectors file" will be trasnferred on to an 8" floppy disk in RT-11 Operating System Data format.

The sub-system (a) will be implemented on VAX-11/780 computer system as the ESIM files are located in that system. This is also favored by the consideration that LSI-11/23 microcomputer has limited (256 kbytes) memory. This will allow the microcomputer memory to be utilized exclusively for sub-system (b).

The sub-system (b), testing of the ICUT, will revolve around a user-friendly interface. The user will be prompted to select a specific mode of operation from the system menus which will be offered at each step of operation. The system will prompt the user to input information regarding size, shape and pin designations of the ICUT. It will then establish cross-reference between ICUT pin numbers and their physical location on the Stanford IC tester. During simulation of the ICUT, the system will translate the test vector bits from ICUT pin-numbers into their physical location on IC tester and provide high/low voltages at these

II. System Design

Overview

The Automated Tester for VLSI circuits (ATV) was basically analyzed using Data Flow Diagrams (Fig. II-1). A data flow diagram is a graphical technique that depicts information flow and the transforms that are applied as data move from input to output [Ref. 2, 98-104]. The physical discontinuity in the system data flow made it logical to sub-divide the overall plan into two sub-systems.

a) Extraction of Test Data from ESIM Files

b) Testing of ICUT (IC Under Test)

The functional requirements of the system, provide the user with an option to operate in "Auto" or "Manual" mode. Operation in "Auto" mode entails that respective ESIM file exists from which test data to simulate the ICUT can be extracted. ESIM files are typically produced during design phase (CAD) of a particular VLSI circuit and contain information regarding node designations and changes in their status values (high or low) during a simulation run. A typical ESIM file, showing its data format and commands, is placed in Appendix "F". The extraction of test data from ESIM file will be achieved by realizing the sub-system (a).

In "Manual" mode of operation, the user will input the test vectors through the keyboard to simulate the ICUT.

The sub-system (a), Extraction of Test Data from ESIM

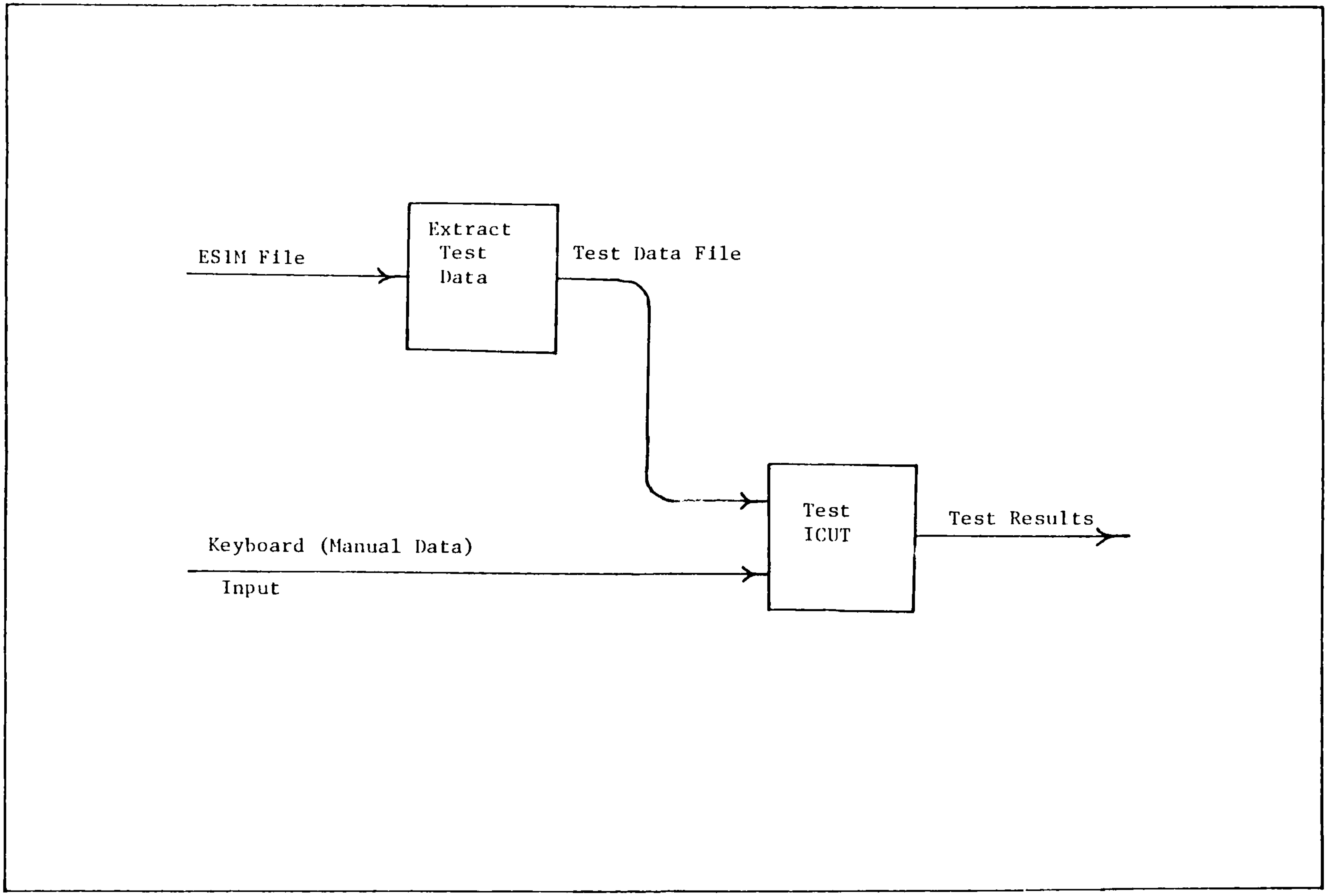


Figure II-2: Functional Decomposition of ATV System

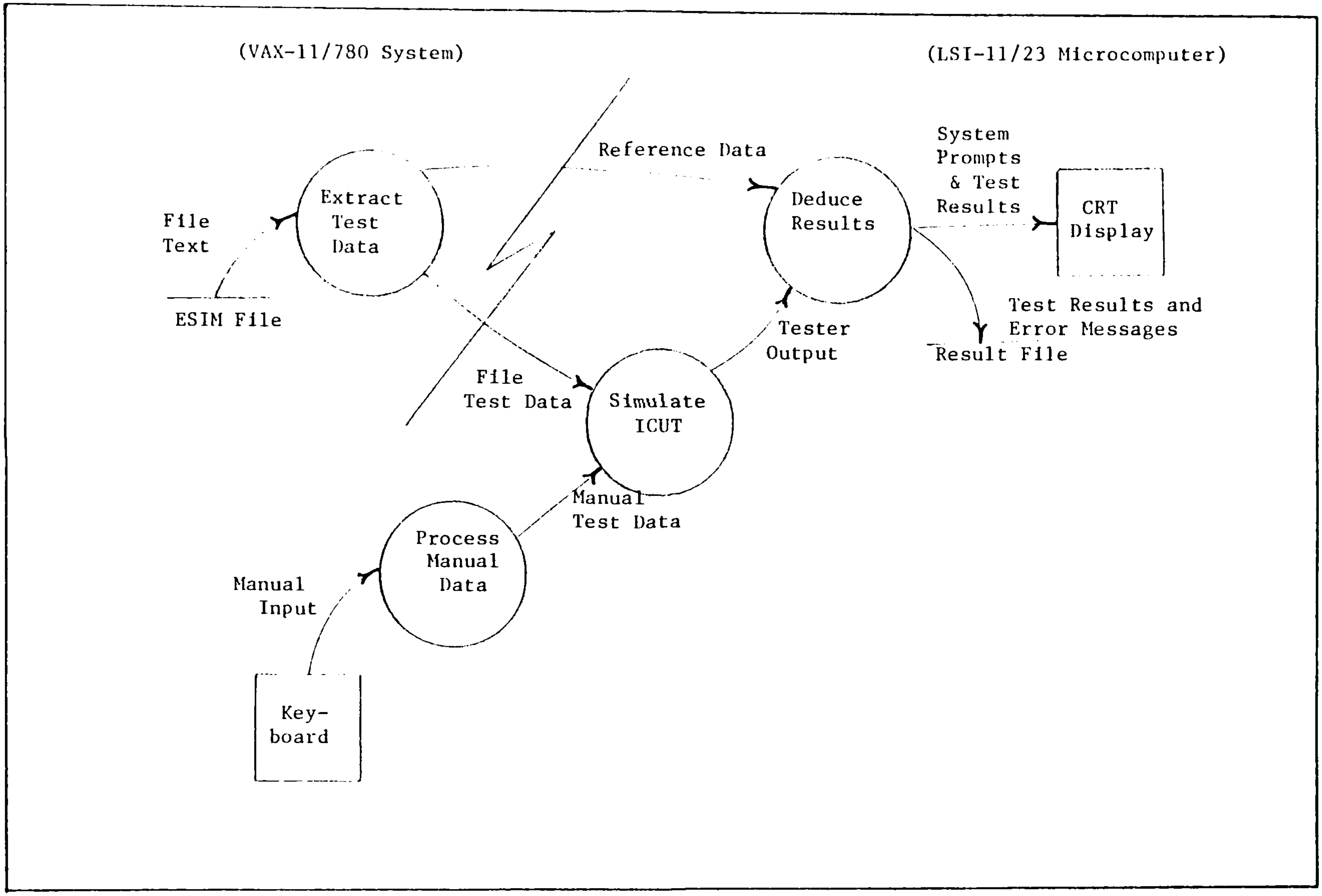


Figure II-1: Data Flow Diagram - ATV System

The most important aspects of the thesis have been, to outline the overall system requirements (Chapter I), System Design (Chapter II), and Analysis (Chapter V). Chapters III and IV consist of detailed design and implementation of two selected functions of the system: "Extract Test Data" and "Process Users' Input". These functions were selected in consultation with thesis advisor, from the complete system design for further elaboration.

Chapter V describes a test plan, which was used to test Extract Test Data function, the implemented part of system design. The remaining part of this chapter analyzes the test results.

Chapter VI summarizes the overall work and contains some recommendations for future work in this field.

application of test data to simulate ICUT, sampling the monitored pins for the output results and comparing these output values with the available reference data to ascertain proper functioning of ICUT.

This part of the system will be implemented on an LSI 11/23 microcomputer. The software developed in this part will reside on floppy disks operable under an RT-11 Operating System.

The ESIM files for a particular VLSI circuit consist of pin-designations of monitored IC-pins, initialization vectors, clocking sequences, test inputs and corresponding output vectors.

The VLSI circuit will be simulated either from test files or from data entered through LSI 11/23 keyboard in an interactive manner. After initialization of IC tester and the ICUT, each test vector will be applied to IC under test and resulting output will be compared with the given reference vector to ascertain proper functioning of the VLSI circuit.

Organization

This report is organized in six chapters: I) Introduction, II) System Design, III) Detailed Design of Selected System Aspects, IV) Implementation of Selected Aspects, V) Analysis, and VI) Conclusion.

(iii) Specification, design, and implementation of menu drivers that will enable the user to interact with the system in a user friendly manner, and,

(iv) Specification, design, and implementation of a program to convert ESIM files produced on a VAX 11/780 computer system into a compatible form for the microcomputer running RT-11 Operating System.

General Approach

The tester system will be controlled by an LSI 11/23 microcomputer with 256 Kbytes of memory. In order to optionally use the limited memory resources of a microcomputer, the basic plan is sub-divided in two parts:

(a) Extraction of Test Data.

This part of the system will scan ESIM files, segregate pertinent test data and then restructure it for subsequent handling by the microcomputer system. This software program will be implemented on a VAX-11/780 computer system. This preprocessing of data on the VAX system would make better use of microcomputer memory available for IC testing programs. The conversion of these data files to RT-11 format will be carried out under a system program available in the UNIX-library.

(b) Testing of ICUT.

Testing of a VLSI circuit includes getting data from the user and the disk, initialization of IC tester and the ICUT, computer driving a hard-wired tester (Stanford IC Tester).

(ii) The ESIM files shall be available on 8" floppy disks in RT-11 Operating System format for use as input files to the tester system.

(iii) Where practical all the software shall be written in the "C" language. The program modules required to be written in "Assembly Language" shall be adequately documented and kept to a minimum.

(iv) The power for the IC tester shall be delivered through the microcomputer chassis.

Performance Requirements

(i) The system software shall not limit the testing capabilities of the Stanford IC Tester.

Limitations

(i) In reference to Functional Requirement iv(b) [Auto Subset Tests], the user has the total responsibility for integrity of test results if out-of-sequence test instructions are run without proper intializations of ICUT.

Scope

The scope of this thesis will be limited to:

(i) Design of the tester system at system level using top-down modular concepts,

(ii) The specification and design of a system controller that will control the microcomputer testing software, output pins of the ICUT and provide power and ground connections to the IC as specified for the test program.

(c) During the third phase (SIMULATION), the system shall read input data vectors from disk and apply them to ICUT in real time. The outputs from the ICUT will be stored in memory and/or on the disk; these results will be compared with pre-stored vectors for fault detection and possibly fault diagnosis.

(vii) The system shall have the following responses for depicting results of a test-run:

(a) It shall display "TEST COMPLETED" in case of successful completion of a test program for an ICUT in "AUTO" mode.

(b) It shall specify the particular input sequence, signal names, and bad pins of the ICUT in case of detecting a fault in "AUTO" mode.

(c) The system shall display the input and output vectors during single-step simulation in "MANUAL" mode.

(viii) The inputs and outputs from the ICUT shall be available for visual inspections on an oscilloscope.

(ix) The input simulation data for any particular ICUT shall be obtained from ESIM-simulator runs performed on any computer system at AFIT (primarily a VAX 11/780).

Implementation Requirements

(i) The system shall consist of a LSI 11/23 micro-

(b) The user shall have the ability to run any set or subset of test instructions.

(c) As a user option, the tester will stop the test either at detecting the first fault or only at the end of the entire test after recording all encountered faults.

(v) The "MANUAL" mode will permit static analysis of the ICUT and shall provide the following options:

(a) The user shall have the ability to manually change the status of any pin of ICUT from low to high and vice versa between any two clock periods.

(b) The user shall have the ability to address individual pins of the ICUT, designate them as input, output, power or clocking pins and provide specific input vectors for simulation of ICUT in single step operation.

(vi) A test run for an ICUT shall consist of the following phases:

(a) During the first phase (GATHER-DATA), pertinent information regarding ICUT, i.e., its specifications, name of test data file, number of pins and their designations and IC-initialization data will be collected and cross reference tables will be generated as a preparatory step to simulate ICUT. This data will be received from user through a user friendly interactive system program.

(b) During the second phase (INITIALIZATION), the system shall initialize the tester, establish the input and

Statement of Problem

The whole project involves developing the hardware interface between the IC tester and a microcomputer, and developing software program which will deduce test sequences from ESIM data files and use them to test VLSI circuits.

Requirements

The requirements for the system are grouped in the following three categories:

(1) Functional Requirements

(2) Implementational Requirements

(3) Performance Requirements

Functional Requirements

(i) The system shall operate and test all ICs within the capabilities of the Stanford IC Tester.

(ii) The system shall be operable through a user friendly menu.

(iii) The system, upon initial start-up, shall give the user an option to operate in "AUTO" or "MANUAL" mode.

(iv) The "AUTO" mode shall provide following sub-options:

(a) The user shall have the ability to run the test program for an IC Under Test (henceforth abbreviated as ICUT) without interruption.

The basic plan of this project is to attach a single-board IC tester developed at Stanford University to a microcomputer and to control the functions of the IC tester with this microcomputer.

Background

This project is an important step toward achieving the AFIT's overall plan for Digital Systems Design Automation. The goal of the AFIT plan has been to take advantage of the rapidly changing field of Design Automation since the future of this field will have a major impact on the future of military technology (Ref. 15).

AFIT has developed a number of operational tools to support its projects and "VLSI design courses". AFIT students have successfully designed a number of "VLSI-Chips" during the past years. These VLSI circuits, after being fabricated, require a simulation test to verify their functional operation. This step was previously done manually, which was far too laborious, monotonous, and often inaccurate.

A tester to simulate the input pins of any Digital Circuit and measure the resultant response at its designated output pins was developed at Stanford University, California (henceforth known as Stanford IC Tester). The overall aim is to utilize Stanford IC Tester, controlled by a microcomputer, to test fabricated VLSI circuits designed at AFIT.

Node A13: Restructure Test Data

Node A131: Store Node Data

Node A132: Change Data Structure

Node A133: Append2 File

Node A14: Change Data Format

Node A2: Process User's Input

Node A21: Classify Input

Node A211: Get Keyboard Input

Node A212: Verify Keyboard Input

Node A22: Select Operating Options

Node A221: Select Mode

Node A222: Verify Mode

Node A223: Setup Options

Node A23: Setup Reference Tables

Node A231: Select Appropriate Pin-Table

Node A232: Fill-In Reference Table

Node A233: Setup Input Pin TAble

Node A234: Setup Clock Pin Table

Node A235: Setup Power/Ground Pin Table

Node A236: Setup Output Pin Table

Node A24: Validate Manual Data

Node A241: Check Overlap with Output Pins

Node A242: Check Overlap with Power/Ground Pins

Node A3: Apply Simulations

- Node A31: Process Test Data
    - Node A311: Segregate Reference Data
    - Node A312: Correlate Tester-Pins & Manual Data
    - Node A313: Correlate Tester-Pins & File Data
    - Node A314: Convert Test Data to SIEVE Format
- Node A32: Perform Test
    - Node A321: Initialize Tester
    - Node A322: Initialize ICUT
    - Node A323: Simulate ICUT
    - Node A324: Sample ICUT Output

Node A4: Deduce Results

- Node A41: Analyze Results
    - Node A411: Convert Sample Data into IC Pin Domain
    - Node A412: Compare Sample Data & Reference Data
- Node A42: Handle Results
    - Node A421: Formulate Test Report
    - Node A422: Generate Storage Buffer
    - Node A423: Copy Buffer to Disk
    - Node A424: Display Test Report

III. Detailed Design of Selected System Aspects

During the preliminary design of Automated Tester for VLSI (ATV), the system was split up into four functional groups as shown in Figure A-2.

These groups are:

i) Extract Test Data

ii) Process Users' Input

iii) Apply Simulation

iv) Deduce Results

The detailed design of the first two groups was undertaken as a part of this thesis. The design of the remaining two groups is left for future development.

Extraction of Test Data from ESIM Files

As described earlier in Chapter II, ESIM files are typically produced during design (CAD) phase of a particular VLSI circuit. These files contain information regarding node designations of a circuit and voltage changes sensed at each node during simulation runs of the VLSI circuit. A typical ESIM file, showing its various commands and data format is placed in Appendix "F".

The function "Extract Test Data" scans a given ESIM file to generate arrays of all monitored pins, input pins and clock pin of a VLSI circuit by interpreting various ESIM commands. A detailed description of all ESIM commands is included in Appendix "F". The ETD function sets up cross-reference between the three arrays (i.e., arrays of monitored pins, input pins and clock pins) and creates an array of output pins from those elements of monitored-pins-array which do not have a cross-reference with arrays of input or clock pins.

The ETD function gathers only pertinent data which is essential for testing a VLSI circuit. This is achieved by activating ETD-function on following, ESIM commands only:

w: list of monitored pins

v: an input node with its respective data vector

k: list of clock pins along with their respective clocking sequence.

I: Initialize ... this is used to establish a cross-reference between array of monitored pins and arrays of input and clock pins. This command is also used to generate array of output pins.

h/l: These commands can change the status of the effected pins. These commands are validated by checking that no output pin is forced to hold a permanently "high/low" voltage status. These commands by holding certain pin to high or low status effect the number of input pins and hence a change in input test vectors. This second part of "h/l" command has not been implemented as yet.

The ETD function generates buffers in memory; each buffer being associated with one particular monitored pin. The function reads node vectors from the given ESIM file and adds it into its respective buffers. If the amount of data exceeds capacity of a buffer, ETD function generates an overflow signal. The overflow signal activates a sub-routine to change the format of available node data into test vectors. These test vectors are written out into an external file that makes the buffers available for any more incoming node data from ESIM file. The change in data format is achieved by visualizing a matrix of ordered input pins (each element being their respective node buffer), and transposing the matrix to generate test vectors.

The ETD function is independent of all other functions and is implemented on VAX-11/780 computer system because all ESIM files are available in that system. This function was considered at system level as narrated in descriptions of Node A1 and its associated child nodes in Chapter II. The preliminary design was expanded to greater depths using SADT diagrams until implementation level.

The system design has been included again in this software phase of detailed design to produce a complete and independent outlook at ETD function.

The node list, node diagrams of the detailed design and their descriptions are attached as Appendix "B". The detailed design was translated into program by use of "C" language. The program listing of ETD function are attached as Appendix "D". The Data Dictionary (Process Definitions and Data Flow Definitions) is included in Appendix "G".

Process Users' Input (PUI)

This function "Process Users Input" actually describes an interactive system interface. The PUI-function prompts the user to select a specific mode of operation from a variety of options (narrated in description of Node A22). For "Auto" mode selection, the PUI function prompts for the name of test data file; it then checks the availability of the given file and for unsuccessful access to the given file, PUI-function changes the mode of operation to "Manual" and informs the user to this effect.

The PUI function asks the user to input information regarding size, shape and pin-designations of the ICUT. For specific shapes and sizes, there are pre-stored tables which correlate the IC pin-numbers to their respective location on Stanford IC tester. The PUI selects the appropriate table and prompts the user to fill in data for all pins, i.e., pin-designations and their class (input, output, clock, power ground, don't care, etc.). The PUI function finally validates input-test vectors in case of "Manual" mode of operation by checking that number of bits do not exceed the number of input pins and no designated output/power or ground pin is included as an input pin.

This function was considered at system level as narrated in the description of Node A2 and its associated child nodes in Chapter II. The system design has been expanded using SADT diagrams to show implementation. This function was translated by use of "C" language and has been partly implemented on LSI-11/23 microcomputer.

The node list, node diagrams of the detailed design and their descriptions are attached as Appendix "C". The Data Dictionary (Process Definitions and Data Flow Definitions) is included in Appendex "G".

IV. Analysis

Test Plan (ETD)

The function "Extract Test Data", like the overall system was designed in top-down manner in accordance with the principles of stepwise refinement which ensured proper integration during later stage of development cycle.

The testing of the ETD function was conducted in two phases. During the first phase, diagnostic testing (to verify correctness of each module) was done mostly by inserting "print-statements". During the second phase, after integration of all modules, a test plan was devised using "equivalence partitioning" technique. [Ref. 4, 44-55]

The primary objective of the test plan was to define a combination of test data that had the highest probability of uncovering errors. "equivalence partitioning" strives to define a test case that uncovers a class of errors that might otherwise require the execution of many test cases before the general error could be observed. This approach greatly reduces the total number of test cases that must be developed.

Statistics have shown that many software errors occur just below, at, or just above the bounding value of indices, data structures and scaler values, and test cases that exercise this domain have a high probability for uncovering errors. [Ref. 2, 289-309]. The test plan was devised using

"Boundry Value Analysis" in conjunction with "Equivalence Partitioning".

This test plan was implemented to check correctness of the overall function (ETD). A basic assumption was made right at the beginning that the user would input the name of a valid ESIM-file, otherwise ETD would try to process the given file and produce "un-intelligent" results.

Identifying the Equivalence Classes

The equivalence classes were identified as listed in Table IV-1. Each external condition represents a class of input data, which was further segregated into sub-class of valid and invalid data-sets. Test cases covering one or more than one valid equivalence classes and one test case for each invalid equivalence class were written down. These test cases were used to evaluate the performance of ETD function.

Test Results

All test cases and cooresponding responses of ETD function are listed in TAble IV-2. ETD function performs as per specification within its scope of implementation.

Limitations

ETD function presently has following limitations which, if removed, would greatly enhance the generality of this function,

(i) implementation of "h" and "l" commands

(ii) implementation of "-w" (unwatch) command.

Table IV-1: Equivalence Classes (Page 1 of 2)

External Conditions	Valid Equivalence Classes	Invalid Equivalence Classes
1) Name of ESIM FIle	- Valid ESIM file Exists (1)	- File Does Not Exist (2) - File Exists but Does Not Contain ESIM Data (3) - No Name is Inputed (4)
2) Name of Storage File	- Valid Name (5)	- No Name (6) - Invalid Name (7)
3) ESIM-File Status	- File Contains Valid Data (8)	- File is Empty (9)
4) Size of ESIM File	- Relatively Small File (1) Size < 4K bytes	- Very Large File (11) Size > 40K bytes
5) Number of Monitored	- > One (12)	- Zero (13) > 40 (14)
6) Number of Input Pins	- > One (15)	- Zero (16) > 20 (17)
7) Number of Output Pins	- > One (18)	- Zero (19) > 20 (20)

Table IV-1: Equivalence Classes (Page 2 of 2)

External Conditions	Valid Equivalence Classes	Invalid Equivalence Classes
8) hl-Command	- Pin Not From Output Pins (22) (23)	- Pin From Output Pins (24) (25)
9) Unwatch (w-pinx) Command (26)	- (Not Implemented)	
10) Repeated Number of (w-watch) Commands in a file (27)	- (Not Implemented)	

Table IV-2: Test Results (ETD) (Page 1 of 3)

Test Case	Result
1. Valid ESIM File, Valid Name (1, 5, 8)	ETD function segregates node data from ESIM files and converts it into test and reference vectors as expected. (See Page IV-8)
2. File Does Not Exist (2)	ETD stops execution after giving a message that "It Is Unable to Open the Said File"
3. File Exists But Does Not Contain ESIM Data (3)	Produces Unintelligent Results
4. No File Name is inputted in Response to System Prompt (CR is Pressed) (4)	ETD Does Not React, It Waits for another input
5. Invalid Name (7) Control Character	ETD stops execution after giving a message that "It is unable to Open the Said File"
6. File is Empty (9)	ETD produces expected results with number of input, clock pin and output pins being zero.
7. Relatively Small File (10)	ETD produces expected results as shown on Page IV-8.
8. Very Large File (11)	ETD function segregates node data from ESIM files and converts it into test and reference vectors as expected.
9. Number of Monitored Pins > One (12)	ETD function segregates node data from ESIM files and converts it into test and reference vectors as expected. (See Page IV-8)

Table IV-2: Test Results (ETD) (Page 2 of 3)

Test Case	Results
10. Number of Monitored Pins Zero (13)	ETD produces an output file with zero output pins and no output vectors (See Page IV-9).
11. Number of Monitored Pins > 40 (14)	The program crashes giving an error message "segmentation fault".
12. Number of Input Pins (15) > One	ETD function segregates node data from ESIM files and converts it into test and reference vectors as expected. (See Page IV-8)
13. Number of Input Pins (16) Zero	ETD produces an output file with number of input pins (zero) and no input vectors (example of such an input and output file is shown on Page IV-10 and IV-11 respectively.)
14. Number of Input Pins (17) > 20	The program crashes giving an error message "segmentation fault".
15. Number of Output Pins (18) > One	ETD function segregates node data from ESIM file and converts it into test and reference vectors as expected. (See Page IV-8).
16. Number of Output Pins (19) Zero	ETD produces an output file with zero output pins and no output vectors. (Example of such an input and output file is shown on Page IV-12.)
17. Number of Output Pins (20) > 20	The program crashes giving an error message "segmentation fault".

Table IV-2: Test Results (ETD) (Page 3 of 3)

Test Case	Result
18. h-Command Effected Pin Being an Output Pin (22)	ETD gives an error message that effected pin is an output pin and respective command is invalid - but it continues with execution of rest of the program.
19. l-Command, Effected Pin Being an Output Pin (23)	"
20. h-Command, Change in status of Effected Pin (24)	Presently a limitation not implemented.
21. l-Command, Change in Status of Effected Pin (25)	"
22. Unwatch Command (26)	"
23. Repeated Number of w-Commands (27)	"

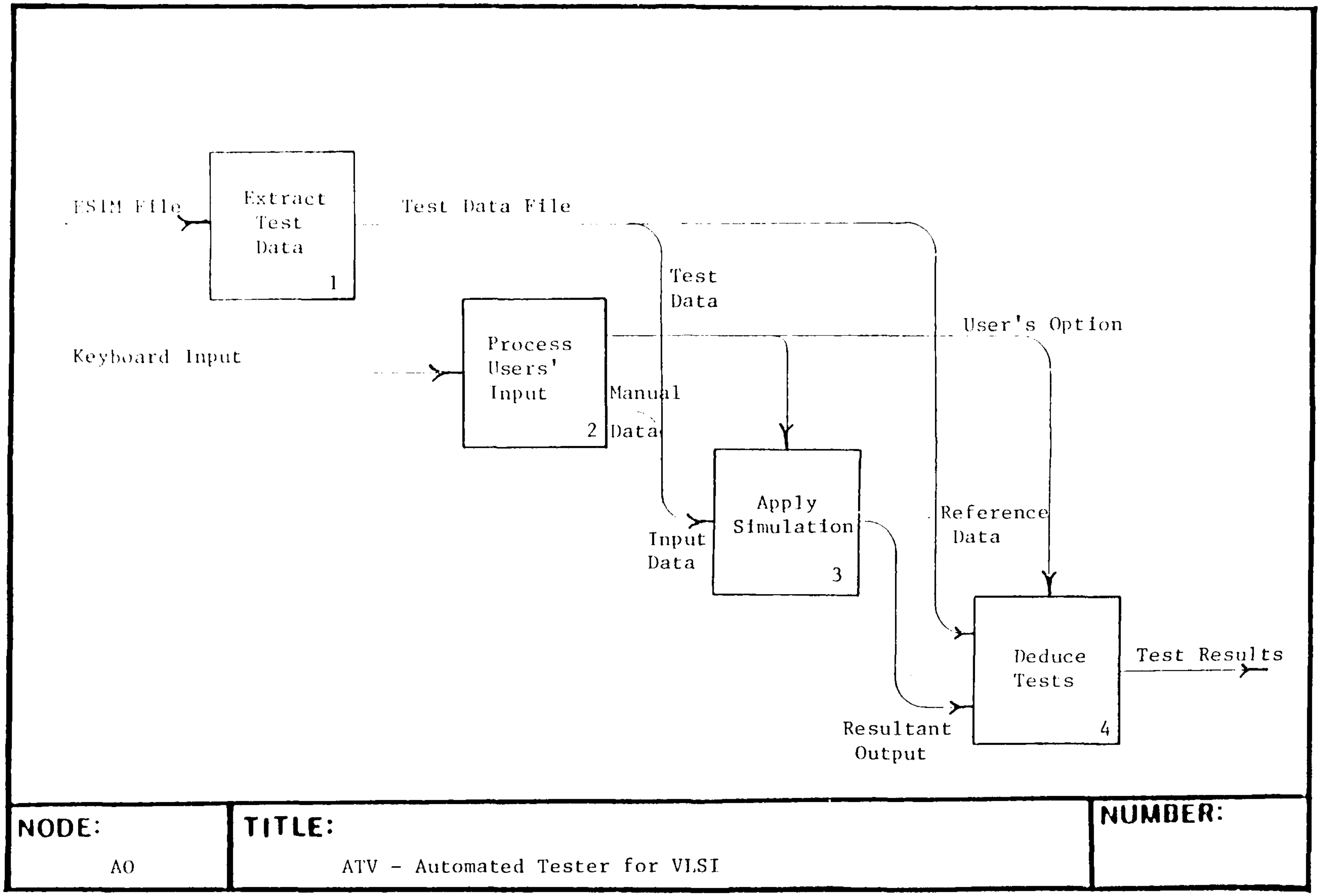


Figure A-2: Automated Tester for VLSI

NODE AO: Automated Tester for VLSI Circuits (ATV)

Abstract

This is the entire system, decomposed into four major functions: Extract Test Data, Process User's Input, Apply Simulations and Deduce Results.

The Tester System (ATV)

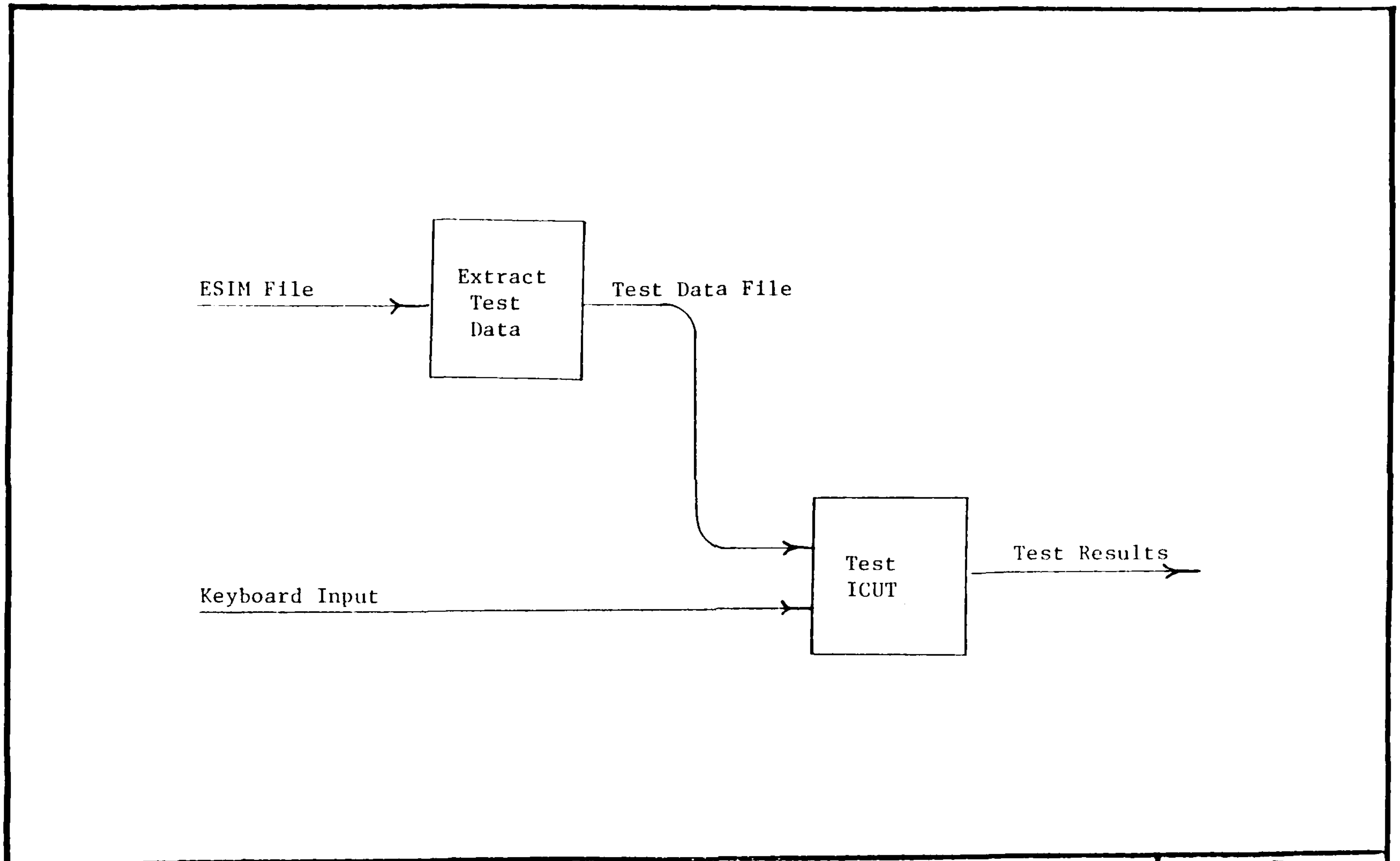
The tester system has been sub-divided into four major functions as shown in Fig. A-2. These functions are:

(i) Extract Test Data: This function scans ESIM file for pertinent test data, segregates test data and restructures the available node data into test vector form for subsequent ease of application. This function is implemented in VAX system where all ESIM files reside.

(ii) Process User's Input: This function prompts the user to select from different operating options: Auto/Manual mode with/without storing test results into a disk file; to input all the testing parameters: name of respective test file in Auto mode, size of IC, its pin-designations and initialization and test data in Manual mode. This function checks the syntax of data received from keyboard and classifies the valid input by setting various flags pertaining to user's operating options.

(iii) Apply Simulations: This function receives test data from keyboard in Manual mode of operation and from test file in Auto mode of operation. It translates test data from IC pin numbers domain to their physical location on tester. This function converts the test data into "SIEVE" format which is particularly required for simulation of Stanford IC Tester.Details of "SIEVE" format are attached as Appendix "E". This function also applies physical "high/low" voltage levels corresponding to a test vector to simulate an ICUT, and samples the tester for resultant output.

(iv) Deduce Results: This function translates the resultant output of IC tester into IC pin numbers domain and compares it with reference data for any inconsistency. This function generates the final test result report from successful/unsuccessful completion of test and pre-stored messages. These test results can be directed to a terminal and/or to a storage file for later consumption as opted by the user.



NODE: A-0

TITLE: ATV - Automated Tester for VLSI

NUMBER:

Figure A-1: Automated Tester for VLSI

Appendix "A"

System Node Descriptions

NODE A-0: Automated Tester for VLSI (ATV)

Abstract

This is the environment node.

ATV-Automated Tester For VLSI

The whole system as shown in Figure A-1 is sub-divided into its two major parts:

(a) extraction of test data
(b) testing of an ICUT (IC Under Test)

The extraction includes reading only pertinent data from the ESIM file and restructuring it from available node data into test vectors for subsequent use; and testing of the ICUT includes simulation of tester by input vectors, detection of tester output, comparison of tester output with the reference data, deduction of results and formulation of test reports in regard to functional capabilities of an ICUT.

Both sub-systems will be implemented in separate environments. The ESIM Files, from which test data has to be extracted, reside in VAX-11/780 system and therefore, software to handle the "data extraction" function will also reside in VAX system. The output of this function "Test Data File" will be available in RT-11 Format on 8" FLoppy disk.

The testing of an ICUT function will be implemented on LSI 11/23 microcomputer which will be used to control the Stanford IC Tester.

At this level, the external input and output as shown in Fig. A-4 are ESIM file, Keyboard input and test results. ESIM files are typically simulator runs for a VLSI IC during its design phase and provide necessary test data to test an IC in "Auto" mode of operation. Keyboard input comprises of all information in respect of IC, its nomenclature, size, name of respective test file, pin designations, mode of operation and testing vectors for "Manual" mode of operation. Test results comprise of information regarding any non-conformity of tester output and reference data and GO/NOGO information for successful/unsuccessful completion of a test. These test results can be directed to a video terminal and/or to a file for storage as opted by the user.

13. Scharer, Laura. "Pinpointing Requirements", Datamation, (pp. 139-151), April 1981.

14. Niederhauser, J. Richard. Digital Logic Simulator, MS Thesis, Wright-Patterson AFB, Ohio. Air Force Institute of Technology, December 1971 (AD 736827).

15. Carter, Harold W. "A Plan for Digital System Design Automation at the Air Force Institute of Technology," Planning Document, Department of Electrical Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, November 1981.

Bibliography

1. Peter, Lawrence J. Software Design: Methods & Techniques, New York: Yourdin Press, 1981.

2. Pressman, Roger S. Software Engineering: A Practitioner's Approach, New York: McGraw-Hill Book Company, 1982.

3. Softech, Inc., "An Introduction to Structured Analysis and Design Technique" (Softech Document #9022-78) November 1976.

4. Myers, G. The Art of Software Testing, New York: Wiley, 1979.

5. Kernighan, Brian W. & Ritchie, Dennis M. The C Programming Language, Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1978.

6. Kochan, Stephen G. Programming in C, New Jersey: Hayden Book Company, Inc. 1983.

7. Plum, Thomas. Learning to Program in C, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

8. Peter, Lawrence J. "Software Representation and Composition Techniques", Proceedings of the IEEE, Vol. 68, No. 9, September 1980.

9. Wirth, Niklaus. "Program Development by Stepwise Refinement" Communications of the ACM, Vol. 14, No. 4, April 1971 (PP221-226).

10. Reifer, Donald J. & Trattner, Stephen. Software Specification Techniques: A Tutorial.

11. Ross, Douglas T. "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Transactions, Vol. SE-3, No. 1, (pp. 16-34), January 1977.

12. Ross, Douglas T. and Schomanji, K. E. "Structured Analysis for Requirement Definition", IEEE Transactions, Vol. SE-3, No. 1, (pp. 6-15), January 1977.

minimal set of reference data from a known functional VLSI circuit to test more circuits of that kind. It would be even better if same thought is extended to Digital PCBs and a Reference Test Data Library is set up. This test station could be a valuable in-house capability at AFIT.

(iii) A prototype system interface (although a little wild!) is in operation which sorts out a part of basic preliminaries to set up test pre-requisites.

This is the present day situation. Stanford IC tester has not been received as yet. It should be procured at its earliest, and its capabilities be tested for best utilization. A clear perception about the capabilities of this IC tester would help in achieving an efficient system as the detailed design and implementation of "actual testing of the ICUT" is yet to be developed.

Recommendations

Following are the recommendations for follow-on thesis work on Automated Tester for VLSI.

(i) Efforts should be directed towards creating a communication protocal and hardware interface between LSI-11/23 microcomputer and Stanford IC tester.

(ii) Another thesis project should be undertaken for expansion of prototype system interface to make it comprehensive and user friendlier. The effort should also be directed towards effecting conversion of test vectors to SIEVE format.

Afterthought

It will be difficult to test those VLSI circuits which were not created at AFIT, or for which respective ESIM files do not exist. Research can be aimed at generating

V. Conclusion

This thesis work consists of a system design of an Automated Tester for VLSI circuits (ATV) and implementation of selected system aspects. The design outlines the testing of an ICUT (IC Under Test) to be carried out in three distinct steps.

(i) Extraction of test data from its respective ESIM file in VAX-11/780 system by using ETD program.

(ii) Converting data format of extracted test data file from VAX-Computer System into RT-11 data format and transferring it on 8" floppy disk. This function is available in an UNIX-System library and is achieved by getting assistance from computer room staff.

(iii) Actual testing of the ICUT on Stanford IC tester controlled by LSI 11/23 microcomputer in an interactive mode.

At present, the following has been achieved:

(i) First step, extraction of test data from ESIM files (with some limitations), has been successfully implemented.

(ii) Second function was aprior available through system library. Same was successfully utilized to change the data format of "extracted test data files" and transfer them on to an 8" floppy disk in RT-11 data format.

Input and ETD Response for Test Case (19)

```
w clock serial_in word_mark wØ w1 w2 w3 tØ t1 t2 t3 t4 t5 reset
V clock Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1

V serial_in ØØ11ØØ11ØØ11ØØ11ØØ11
V word_mark ØØØØØØ11ØØØØØØØØØØ11
V wØ ØØØØØØ111111ØØØØ1
V w1 ØØØØØØ111111ØØØØ1
V w2 ØØØØØØ111111ØØØØ1
V w3 ØØØØØØ111111ØØØØ1
V tØ ØØØØØØ111111ØØØØ1
V t1 ØØØØØØ111111ØØØØ1
V t2 ØØØØØØ111111ØØØØ1
V t3 ØØØØØØ111111ØØØØ1
V t4 ØØØØØØ111111ØØØØ1
V t5 ØØØØØØ111111ØØØØ1
V reset 11ØØØØØØØØØØØØØØØØØØ

I
R
>ØØ11ØØ11ØØ11ØØ11ØØ11:serial_in
>ØØØØØØ11ØØØØØØØØØØ11:word_mark
>ØØØØØØØØØØØØØØØØØØØØ:wØ
>ØØØØØØØØØØØØØØØØØØØØ:w1
>ØØØØØØØØØØØØØØØØØØØØ:w2
>ØØØØØØØØØØØØØØØØØØØØ:w3
>ØØØØØØØØØØØØØØØØØØØØ:tØ
>ØØØØØØØØØØØØØØØØØØØØ:t1
>ØØØØØØØØØØØØØØØØØØØØ:t2
>ØØØØØØØØØØØØØØØØØØØØ:t3
>ØØØØØØØØØØØØØØØØØØØØ:t4
>ØØØØØØØØØØØØØØØØØØØØ:t5
 11ØØØØØØØØØØØØØØØØØØ:reset
pulled up transistors(12)
```

```
 INPIN     14      clock serial_in word_mark wØ w1 w2 w3 tØ t1 t2 t3 t4 t5 reset
 CLKPIN    Ø
 OUTPIN    Ø
ØØØØØØØØØØØØØ1                                1     Ø
1ØØØØØØØØØØØØ1                                1     Ø
Ø1ØØØØØØØØØØØØ                                1     Ø
11ØØØØØØØØØØØØ                                1     Ø
ØØØØØØØØØØØØØØ                                1     Ø
1ØØØØØØØØØØØØØ                                1     Ø
Ø111111111111Ø                                1     Ø
1111111111111Ø                                1     Ø
ØØØ1111111111Ø                                1     Ø
1ØØ1111111111Ø                                1     Ø
Ø1Ø1111111111Ø                                1     Ø
11Ø1111111111Ø                                1     Ø
ØØØØØØØØØØØØØØ                                1     Ø
1ØØØØØØØØØØØØØ                                1     Ø
Ø1ØØØØØØØØØØØØ                                1     Ø
11ØØØØØØØØØØØØ                                1     Ø
ØØØ1111111111Ø                                1     Ø
1ØØ1111111111Ø                                1     Ø
Ø111111111111Ø                                1     Ø
1111111111111Ø                                Ø     Ø
```

ETD Response for Test Case (16)

```
 INPIN     Ø
 CLKPIN    Ø
 OUTPIN    13    clock serial_in word_mark wØ w1 w2 w3 tØ t1 t2 t3 t4 t5
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø1ØØØØØØØØØØØØ           1     Ø
>                    11ØØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø11ØØØØØØØØØØØ           1     Ø
>                    111ØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø1ØØØØØØØØØØØØ           1     Ø
>                    11ØØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø1ØØØØØØØØØØØØ           1     Ø
>                    11ØØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø11ØØØØØØØØØØØ           1     Ø
>                    111ØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø1ØØØØØØØØØØØØ           1     Ø
>                    11ØØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø11ØØØØØØØØØØØ           1     Ø
>                    111ØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø1ØØØØØØØØØØØØ           1     Ø
>                    11ØØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø1ØØØØØØØØØØØØ           1     Ø
>                    11ØØØØØØØØØØØØ           1     Ø
>                    ØØØØØØØØØØØØØØ           1     Ø
>                    1ØØØØØØØØØØØØØ           1     Ø
>                    Ø11ØØØØØØØØØØØ           1     Ø
>                    111ØØØØØØØØØØØ           Ø     Ø
```

Input for Test Case (16)

```
w clock serial_in word_mark wØ w1 w2 w3 tØ t1 t2 t3 t4 t5
; fret
I
p
456 transistors, 22Ø nodes (Ø pulled up)
initialization took 285 steps
t5=X t4=X t3=X t2=X t1=X tØ=X w3=X w2=X w1=X wØ=X word_mark=X serial_in=X
clock=X
h inputs: Vdd gnd
l inputs: GND vdd reset
t5=Ø t4=Ø t3=Ø t2=Ø t1=Ø tØ=Ø w3=Ø w2=Ø w1=Ø wØ=Ø word_mark=1 serial_in=1
clock=1
h inputs: Vdd gnd clock serial_in word_mark
l inputs: GND vdd reset
ØØ1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1:clock
ØØØ11ØØ11ØØ11ØØ11ØØ11:serial_in
ØØØØØØØ11ØØØØØØØØØØ11:word_mark
ØØØØØØØØØØØØØØØØØØØØØ:wØ
ØØØØØØØØØØØØØØØØØØØØØ:w1
ØØØØØØØØØØØØØØØØØØØØØ:w2
ØØØØØØØØØØØØØØØØØØØØØ:w3
ØØØØØØØØØØØØØØØØØØØØØ:tØ
ØØØØØØØØØØØØØØØØØØØØØ:t1
ØØØØØØØØØØØØØØØØØØØØØ:t2
ØØØØØØØØØØØØØØØØØØØØØ:t3
ØØØØØØØØØØØØØØØØØØØØØ:t4
ØØØØØØØØØØØØØØØØØØØØØ:t5
t5=Ø t4=Ø t3=Ø t2=Ø t1=Ø tØ=Ø w3=Ø w2=Ø w1=Ø wØ=Ø word_mark=1 serial_in=1
clock=1
h inputs: Vdd gnd clock serial_in word_mark
l inputs: GND vdd reset
456 transistors, 22Ø nodes (Ø pulled up)
h k2 k3
ØØ1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1:clock
ØØØ11ØØ11ØØ11ØØ11ØØ11:serial_in
ØØØØØØØ11ØØØØØØØØØØ11:word_mark
ØØØØØØØØØØØØØØØØØØØØØ:wØ
ØØØØØØØØØØØØØØØØØØØØØ:w1
ØØØØØØØØØØØØØØØØØØØØØ:w2
ØØØØØØØØØØØØØØØØØØØØØ:w3
ØØØØØØØØØØØØØØØØØØØØØ:tØ
ØØØØØØØØØØØØØØØØØØØØØ:t1
ØØØØØØØØØØØØØØØØØØØØØ:t2
ØØØØØØØØØØØØØØØØØØØØØ:t3
ØØØØØØØØØØØØØØØØØØØØØ:t4
ØØØØØØØØØØØØØØØØØØØØØ:t5
pulled up transistors(12)
```

Input and ETD Response for Test Case (10)

```
w
V clock Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1

V serial_in ØØ11ØØ11ØØ11ØØ11ØØ11
V word_mark ØØØØØØ11ØØØØØØØØØØ11
V wØ ØØØØØØ111111ØØØØ1
V w1 ØØØØØØ111111ØØØØ1
V w2 ØØØØØØ111111ØØØØ1
V w3 ØØØØØØ111111ØØØØ1
V tØ ØØØØØØ111111ØØØØ1
V t1 ØØØØØØ111111ØØØØ1
V t2 ØØØØØØ111111ØØØØ1
V t3 ØØØØØØ111111ØØØØ1
V t4 ØØØØØØ111111ØØØØ1
V t5 ØØØØØØ111111ØØØØ1
V reset 11ØØØØØØØØØØØØØØØØØØ

:
P
 ØØ11ØØ11ØØ11ØØ11ØØ11:serial_in
 ØØØØØØ11ØØØØØØØØØØ11:word_mark
 ØØØØØØØØØØØØØØØØØØØØ:wØ
 ØØØØØØØØØØØØØØØØØØØØ:w1
 ØØØØØØØØØØØØØØØØØØØØ:w2
 ØØØØØØØØØØØØØØØØØØØØ:w3
 ØØØØØØØØØØØØØØØØØØØØ:tØ
 ØØØØØØØØØØØØØØØØØØØØ:t1
 ØØØØØØØØØØØØØØØØØØØØ:t2
 ØØØØØØØØØØØØØØØØØØØØ:t3
 ØØØØØØØØØØØØØØØØØØØØ:t4
 ØØØØØØØØØØØØØØØØØØØØ:t5
pulled up transistors(12)
```

```
 INPIN     14      clock serial_in word_mark wØ w1 w2 w3 tØ t1 t2 t3 t4 t5 reset
 CLKPIN    Ø
 OUTPIN    2
 ØØØØØØØØØØØØØ1                                1     Ø
 ØØØØØØØØØØØØØ1                                1     Ø
 1ØØØØØØØØØØØØØ                                1     Ø
 1ØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 11ØØØØØØØØØØØØ                                1     Ø
 11ØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 1ØØØØØØØØØØØØØ                                1     Ø
 1ØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 1ØØØØØØØØØØØØØ                                1     Ø
 1ØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 ØØØØØØØØØØØØØØ                                1     Ø
 11ØØØØØØØØØØØØ                                1     Ø
 11ØØØØØØØØØØØØ                                Ø     Ø
```

Input and ETD Response for Test Cases (1, 5, 8, 10, 15, 18)

```
w clock serial_in word_mark wØ w1 w2 w3 tØ t1 t2 t3 t4 t5
V clock Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1
V serial_in ØØ11ØØ11ØØ11ØØ11ØØ11
V word_mark ØØØØØØ11ØØØØØØØØØØ11
K clock ØØ1Ø phi1 Ø1ØØ phi2 1Ø1Ø
V reset 11ØØØØØØØØØØØØØØØØØØ
V set   ØØØØØØØØØØØØØØØØØØ11
I
R
456 transistors, 22Ø nodes (Ø pulled up)
initialization took 285 steps
t5=X t4=X t3=X t2=X t1=X tØ=X w3=X w2=X w1=X wØ=X word_mark=X
serial_in=X clock=X
h inputs: Vdd gnd
l inputs: GND vdd reset
>Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1:clock
>ØØ11ØØ11ØØ11ØØ11ØØ11:serial_in
>ØØØØØØ11ØØØØØØØØØØ11:word_mark
>Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1ØØ11:wØ
>ØØ11ØØ11ØØ11ØØ11ØØ11:w1
>ØØØØ11ØØØØ11ØØØØ1Ø11:w2
>1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø11:w3
>11Ø11Ø11Ø11Ø11Ø11Ø11:tØ
>11ØØ11ØØ11ØØ11ØØØØ11:t1
>1ØØ1ØØ1ØØ1ØØ1ØØ1ØØ11:t2
>1ØØØ1ØØØ1ØØØ1ØØØ1Ø11:t3
>111ØØØ111ØØØ111ØØØ11:t4
>1ØØØØØØØØØØØØØØØ1Ø11:t5
t5=Ø t4=Ø t3=Ø t2=Ø t1=Ø tØ=Ø w3=Ø w2=Ø w1=Ø wØ=Ø word_mark=1
serial_in=1
clock=1
h inputs: Vdd gnd clock serial_in word_mark
l inputs: GND vdd reset
456 transistors, 22Ø nodes (Ø pulled up)
```

```
 INPIN      5       clock serial_in word_mark reset set
 CLKPIN     3       clock phi1 phi2
      CLK1   clock      ØØ1Ø
      CLK2   phi1       Ø1ØØ
      CLK3   phi2       1Ø1Ø
 OUTPIN     1Ø      wØ w1 w2 w3 tØ t1 t2 t3 t4 t5
>ØØØ1Ø              ØØØ1111111                  1       Ø
>1ØØ1Ø              1ØØØ11ØØ1Ø                  1       Ø
>Ø1ØØØ              Ø1Ø1ØØØØ1Ø                  1       Ø
>11ØØØ              11ØØ1Ø1ØØØ                  1       Ø
>ØØØØØ              ØØ1111Ø1ØØ                  1       Ø
>1ØØØØ              1Ø1ØØ1ØØØØ                  1       Ø
>Ø11ØØ              Ø1Ø11Ø1Ø1Ø                  1       Ø
>111ØØ              11ØØ1ØØØ1Ø                  1       Ø
>ØØØØØ              ØØØ1Ø1Ø11Ø                  1       Ø
>1ØØØØ              1ØØØ111ØØØ                  1       Ø
>Ø1ØØØ              Ø1111ØØØØØ                  1       Ø
>11ØØØ              111ØØØØØØØ                  1       Ø
>ØØØØØ              ØØØ1111110                  1       Ø
>1ØØØØ              1ØØØ11ØØ1Ø                  1       Ø
>Ø1ØØØ              Ø1Ø1ØØØØ1Ø                  1       Ø
>11ØØØ              11ØØ1Ø1ØØØ                  1       Ø
>ØØØØØ              ØØ111ØØ1Ø1                  1       Ø
>1ØØØØ              ØØØØØØØØØØ                  1       Ø
>Ø11Ø1              1111111111                  1       Ø
>111Ø1              1111111111                  Ø       Ø
```

NODE A1: Extract Test Data

Abstract

This node scans the ESIM file to separate out pertinent test data. It classifies the extracted data into test input data and reference output data. This function restructures this data available in node configuration into vector form. It also changes the memory storage pattern of restructured test data from VAX computer system to RT-11 Operating System data format.

Extract Test Data

This node is decomposed into four major functions as shown in Fig. A-3. These functions are:

(i) Read ESIM File: This function reads in the given ESIM File line by line and classifies each incoming line into command-line or data-line. A brief explanation of all commands encountered in ESIM files is attached as Appendix "F".

(ii) Tabulate Command Data: This function interprets the commands in a given command-line to setup different arrays to contain names of all monitored pins, input pins and clock pins. This function also generates an array of output pins from the already available pin lists.

(iii) Restructure Test Data: This function interprets the data line and stores all the test data pertaining to a node into its respective buffer. This function does some housekeeping by converting the node data into vector form and storing it in an external file to empty buffer for the following node data. This is done when a change in pin-designation is made or data buffer overflows due to incoming data.

(iv) Change Data Format: This function changes the memory storage pattern of restructured test data from VAX computer system into RT-11 data format. This function is implemented by a system library routine and therefore, it will not be elaborated any further.

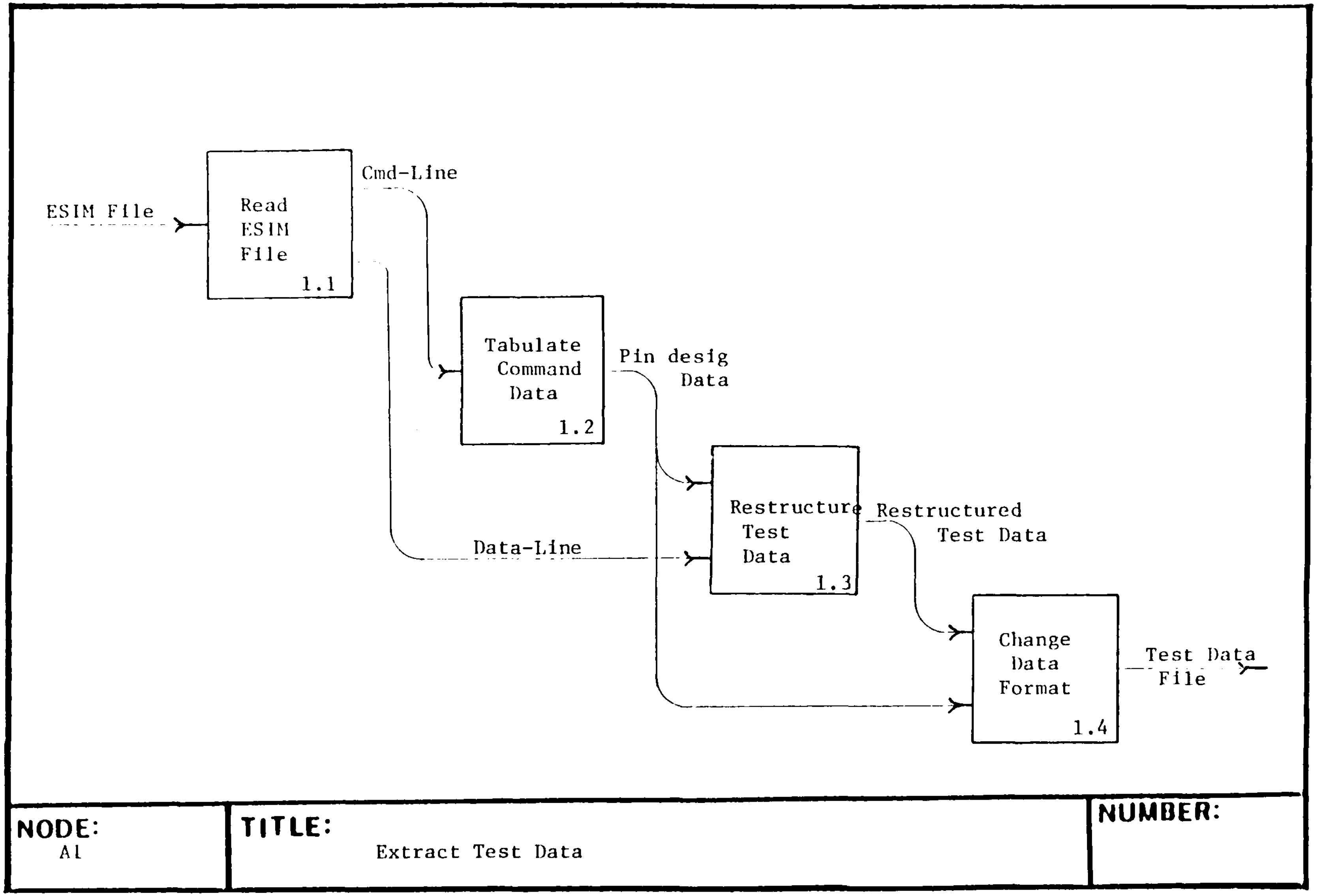


Figure A-3: Extract Test Data

NODE All: Read ESIM File

Abstract

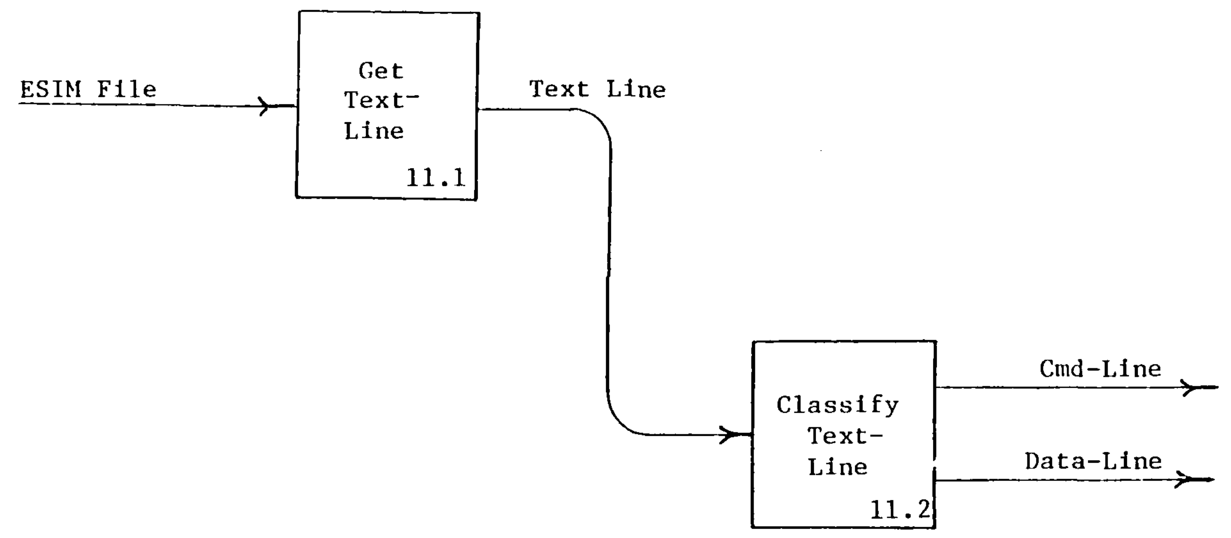
This node reads in given ESIM file, line by line until the end of file is reached and classifies each incoming line to be either a command-line or a data-line.

Read ESIM File

This node is decomposed into two functions as shown in Fig. A-4. These functions are:

(i) Get Textline: This function reads a given ESIM file line by line until the end of file and is implemented by a system library routine (fgets). This function is not elaborated any further.

(ii) Classify Textline: This function, on receiving a text-line from ESIM file categorizes it to be either a command line if the first character of the textline is "alphabetic" or to be a dataline if the first character of given textline is ">". A brief explanation of terminology used in ESIM files is attached as Appendix "F".



NODE: A-11	**TITLE:** Read ESIM-File	**NUMBER:**

Figure A-4: Read ESIM File

Node A12: Tabulate Command Data

Abstract

This node interprets the command in a given command line to setup arrays of monitored pins, input pins and clock pins. This function generates an array of output pins from available pin-data and also changes the status of any designated pin on receiving specific instructions.

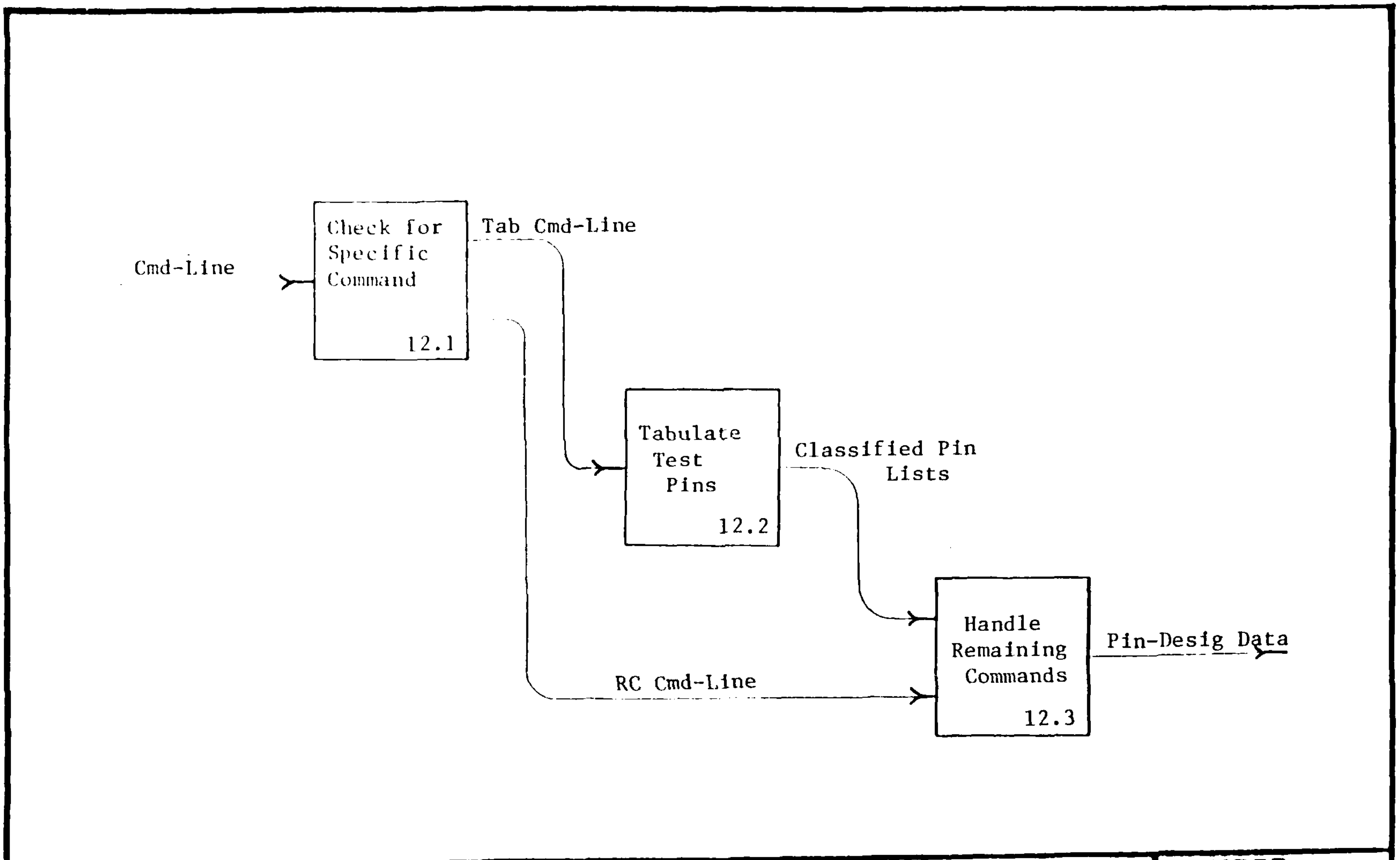
Tabulate Command Data

This node is decomposed into three major functions as shown in Fig. A-5. These functions are:

(i) Check for Specific Command: This function singles out the first character of a given command line and branches out to perform the specific sub-function on its interpetation.

(ii) Tabulate Test Pins: This function generates arrays of monitored pins, input pins and clock pins. It establishes reference between all three arrays. This function also stores available initialization data and clock sequences and establishes correspondence between this data and related pins.

(iii) Handle Remaining Commands: This function, on interpreting the first character of command line to be "h or l", scans arrays of output pins and clock pins to check if any pin from these categories has not been forced to hold a status of "high/low" voltage level. This function changes the status of other pins outside above two categories.



NODE:	TITLE:	NUMBER:
A12	Tabulate Command Data	

Figure A-5: Tabulate Command Data

NODE A13: Restructure Test Data

Abstract

This node interprets the data line and stores all the test data pertaining to a node into its respective buffer. This node does some house keeping by converting the node data into vector form and storing it in an external file to empty buffer for the following node data. This is done when a change in pin-designation is made or data buffer overflows due to incoming data.

Restructure Test Data

This node is decomposed into four major functions as shown in Fig. A-6. These functions are:

(i) Store Node Data: This function on receiving a data line, interprets the specific node for which data line is intended and add the data to a buffer attached to that node. It generates an overflow error if the incoming data exceeds the capacity of respective buffer.

(ii) Change Data Structure: This function converts the available test data in node configuration into vector form. It changes the node data related to input pins into test vectors and node data related to output pins into reference vectors.

(iii) Append2-File: This function empties the buffer by writing all restructured data to an external file, whose name has been provided by the user.

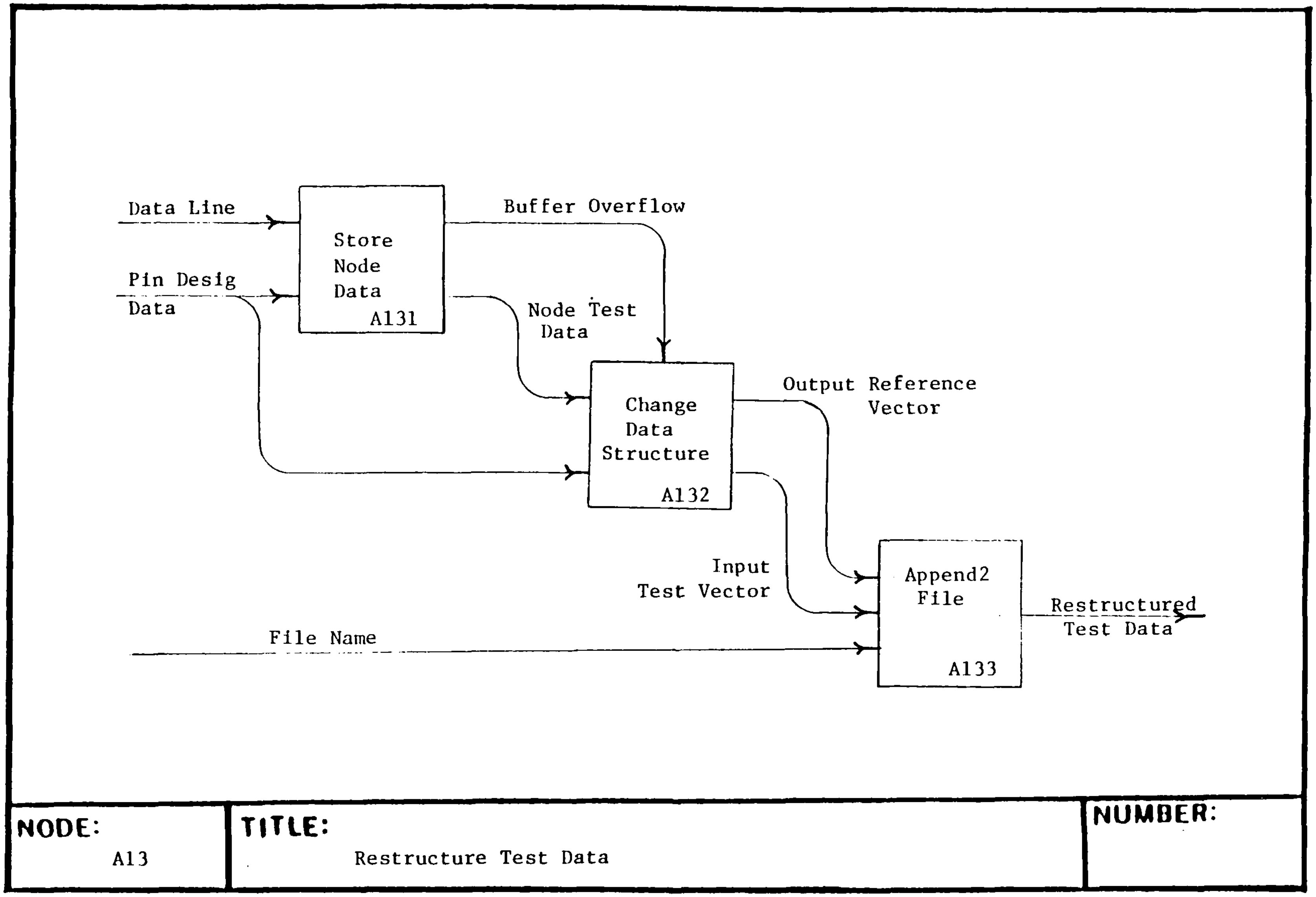


Figure A-6: Restructure Test Data

NODE A14: Change Data Format

Abstract

This node changes the memory storage pattern of restructured test data from VAX computer system into RT-11 Operating System data format. This node is implemented by a system library routine and therefore, it will not be elaborated any further. (Fig. A-7)

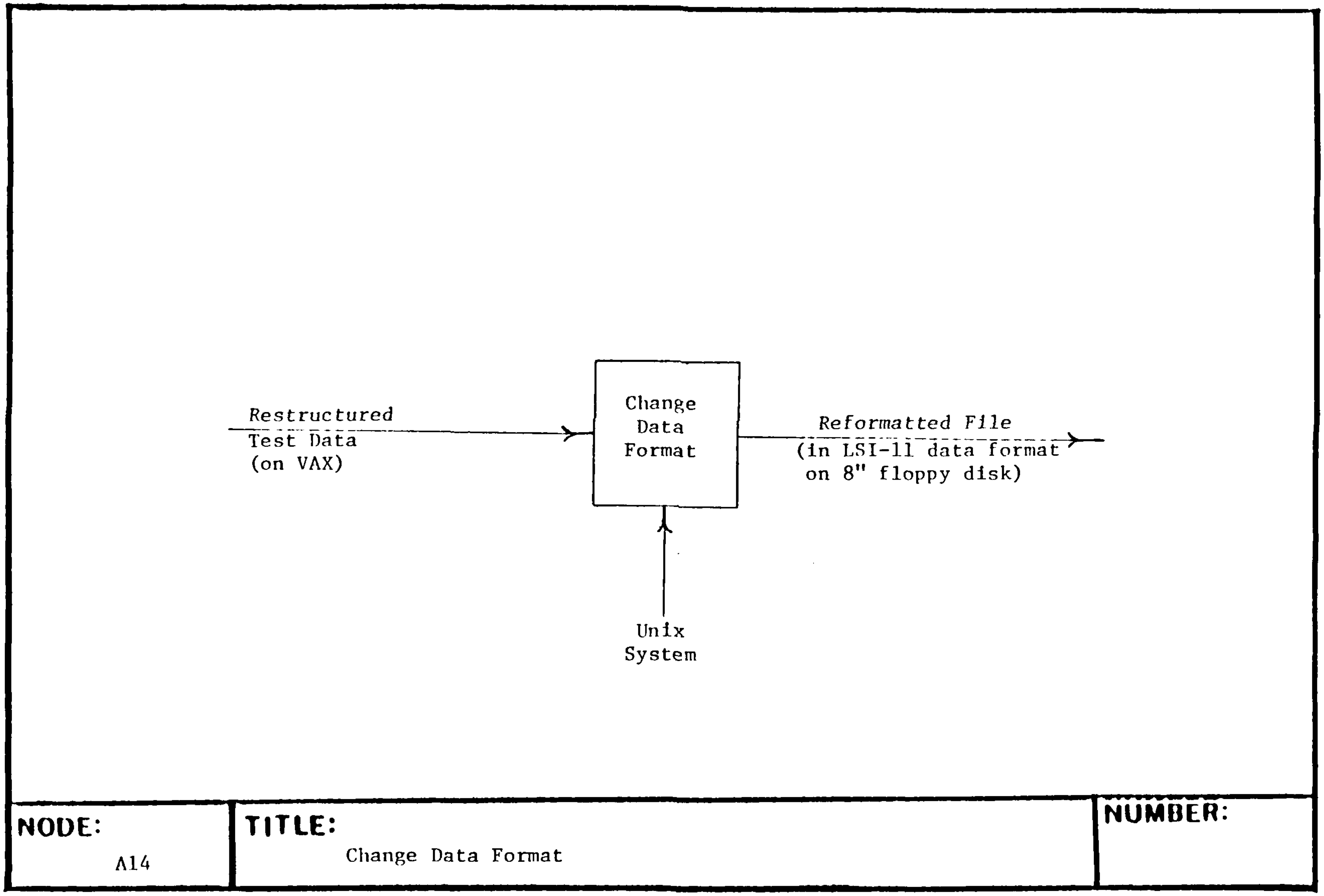


Figure A-7: Change Data Format

NODE A2: Process User's Input

Abstract

This node on receiving all command/data input by the user in response to system prompts, checks its syntax, and sets various flags for operating mode conditions; it sets up reference tables to correlate IC pin numbers with their physical location on IC Tester and validates test data for "Manual" mode of operation.

Process User's Input

This node is decomposed into four major functions as shown in Fig. A-8. These functions are:

(i) Classify Input: This function receives all characters or character strings from keyboard as command or data input in response to system prompt and classifies all inputs into three broad categories, IC data, Option Data and Test Data. It also sets various flags under system prompt to aid in classification of incoming data.

(ii) Select Operating Options: This function allows the user to operate in any desired mode from a range of selections offered in system menues. This function sets different flags to run the program according to user's option.

(iii) Setup Reference Tables: This function selects one of the prestored tables depending on physical characteristics of ICUT. These reference tables contain information to set one to one correspondence between IC pins and IC tester pins for an IC of particular physical characteristics. It then prompts the user to provide information about input/output and other significant pins of ICUT. This function stores this data to correlate IC pin numbers in a test vector to their physical location on IC Tester for effecting a test simulation.

(iv) Validate Manual Data: This function is activated only in manual mode of operation. It checks for any overlap of input test data over designated output/power/ground pins.

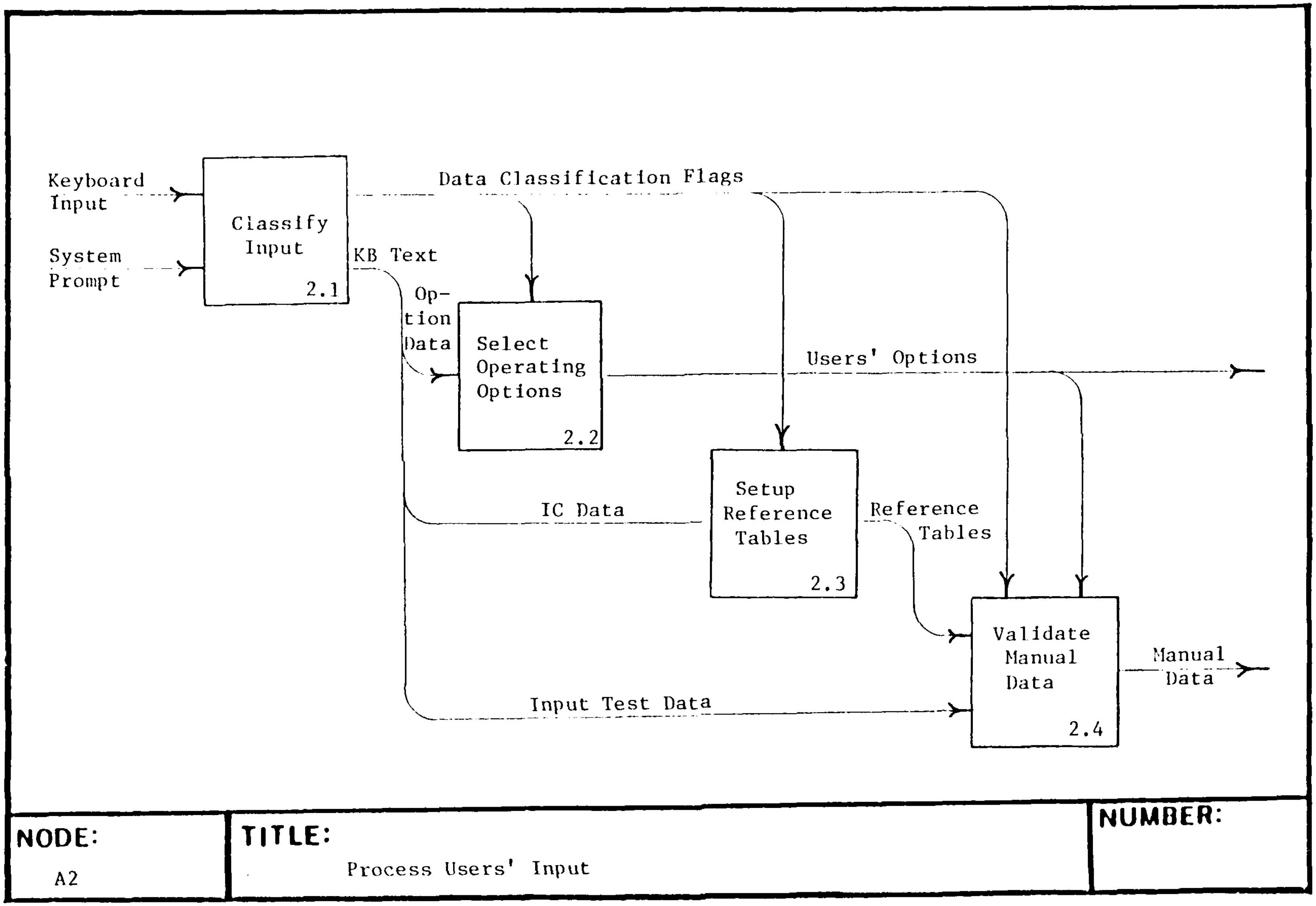


Figure A-8: Process Users' Input

NODE A21: Classify Input

Abstract

This node receives all command/data input from keyboard, checks its syntax, and categorizes the input into three broad classes: ICdata, Option Data, and Test Data.

Classify Input

This node is decomposed into two major functions as shown in Fig. A-9. These functions are:

(i) Get Keyboard Input: This function scans input port and gets any character or character string input from keyboard in response to system prompt. This function is implemented through routines available in system library, and therefore, it will not be elaborated any further.

(ii) Verify Keyboard Input: This function sets one of three data classification "flags" from system prompt to classify the input inthree broad categories of IC Data, Option Data and Test Data for further processing.

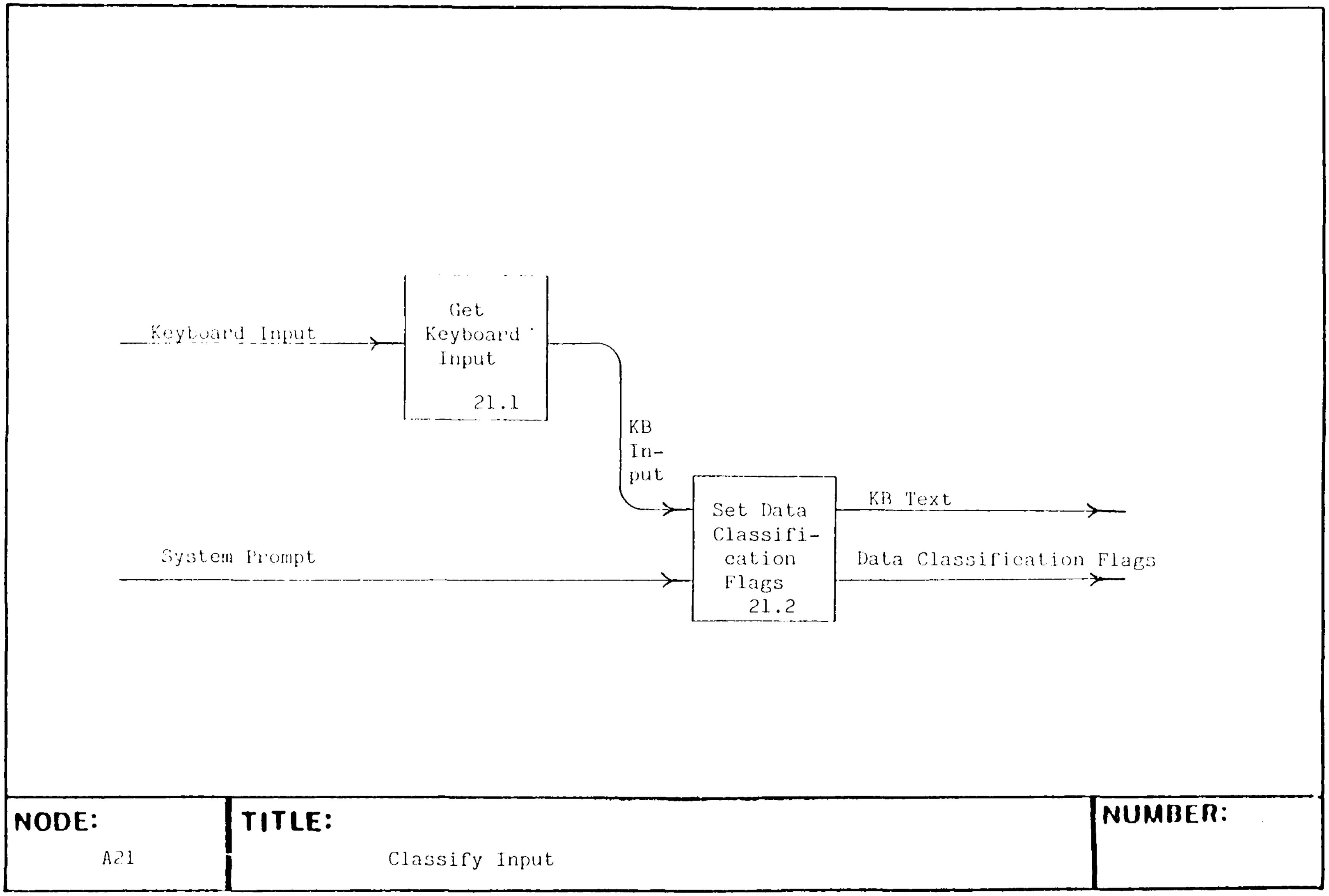


Figure A-9:   Classify Input

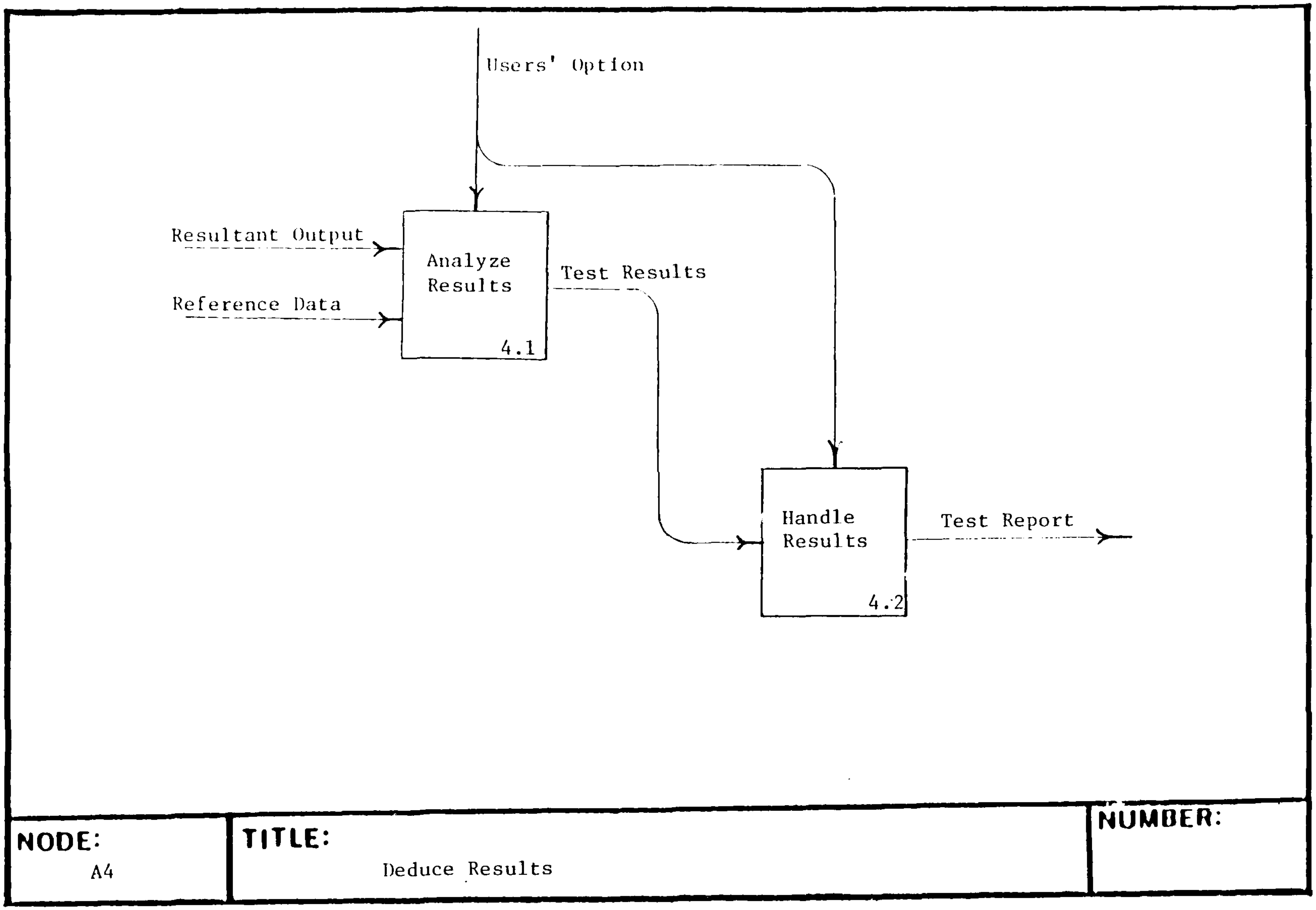


Figure A-16: Deduce Results

NODE A4: Deduce Results

Abstract

This node converts the ICUT output into IC pin domain and analyzes this output by comparing it with the reference data. This node also transforms the analysis results into test report for user's consumption.

Deduce Results

This node is decomposed into two major functions as shown in Fig. A-16. These functions are:

(i) Analyze Results: This function transforms the sampled output of IC tester into ICUT pin numbers' domain by consulting reference tables. It segregates the output vector from total output and compares it with the reference data to single out any mismatch and point of mismatch. It also generates an additional message on successful/unsuccessful completion of test.

(ii) Handle Results: This function, on receiving a test result, maps it into a test report from pre-stored messages. This function depending on users' option allows display of test results on terminal and/or storing these test reports in a file for later consumption.

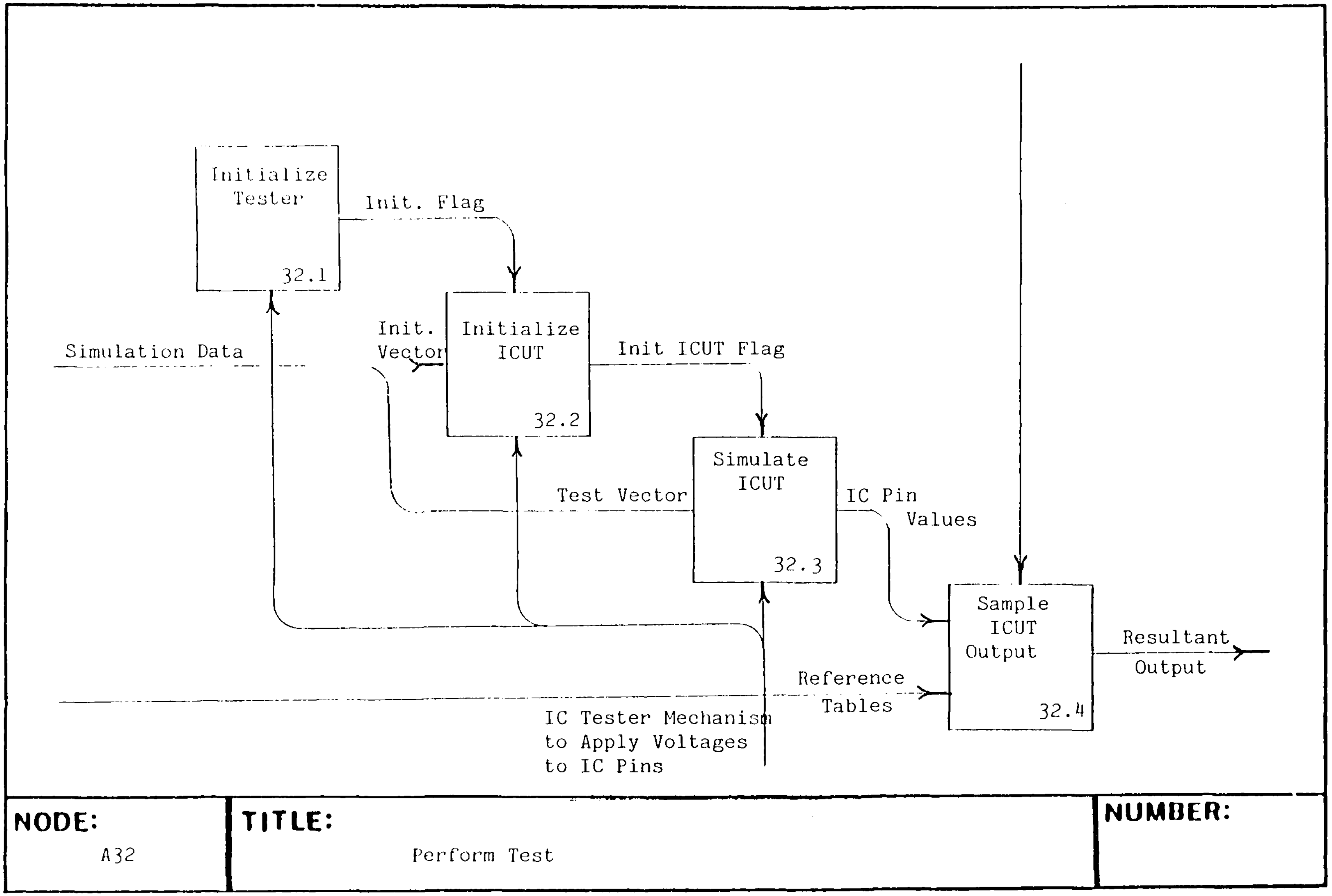


Figure A-15: Perform Test

NODE A32: Perform Test

Abstract

This node performs actual testing of ICUT (IC Under Test). Testing includes initialization of IC tester and ICUT, simulating ICUT with a test vector and sampling its ouput for evaluation.

Perform Test

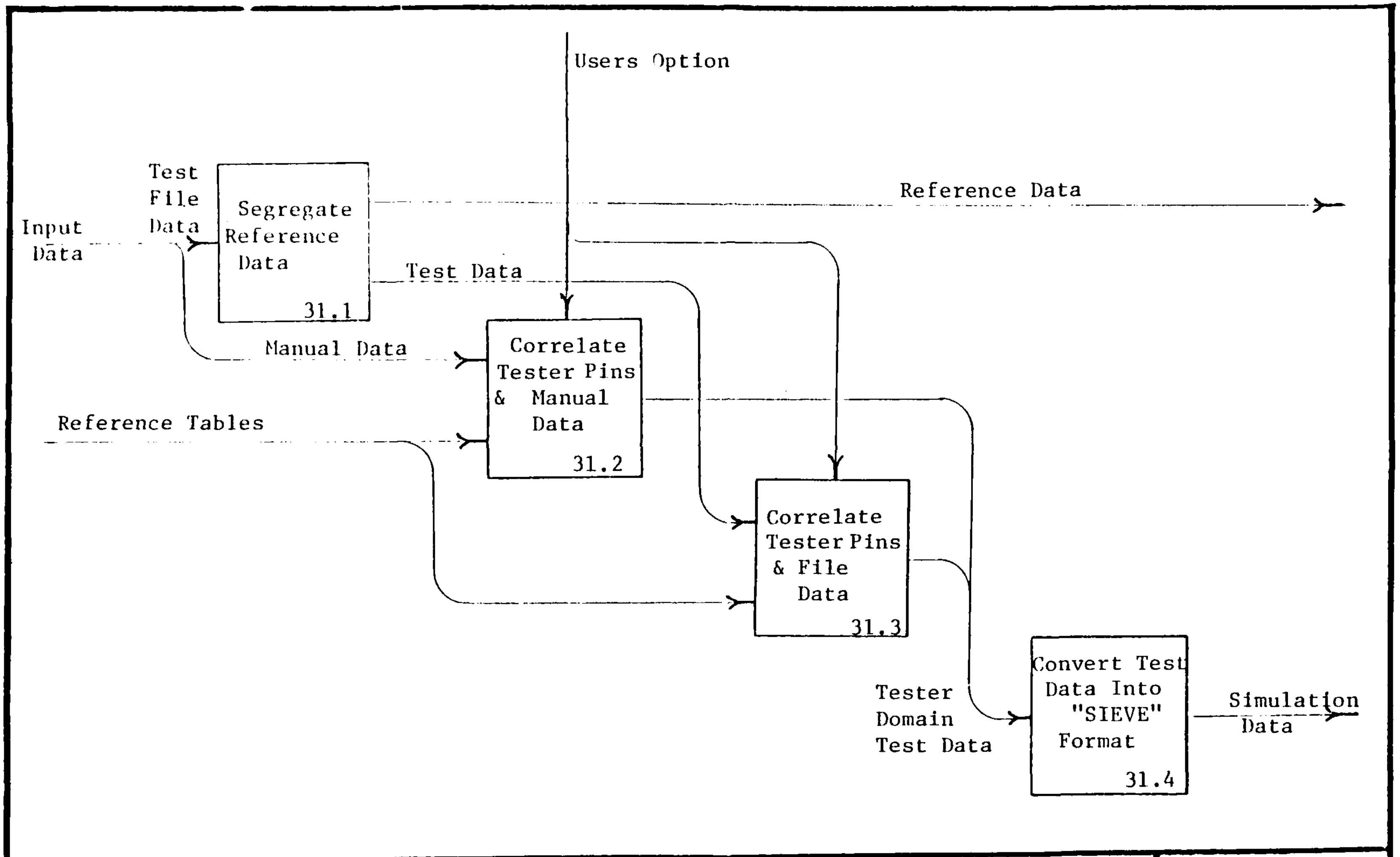
This node has been decomposed into four major functions as shown in Fig. A-15. These functions are:

(i) Initialize Tester: This function physically applies "Ground/Power" voltages to pertinent tester pins to force them out of ambiguous logic states.

(ii) Initialize ICUT: This function simulates ICUT with users' supplied data or initialization data from test data file (ESIM file) through a predetermined number of clock cycles to force the status of output pins to a steady state value. The number of clock cycles are determined by the available data.

(iii) Simulate ICUT: This function after being supplied with a test vector and a flag to proceed, applies corresponding voltages to pins of an ICUT. This function also translates the clocking sequence into voltage variation at required pin to provide a simulation to the IC under test.

(iv) Sample ICUT: This function stores the status of all pins of ICUT after it has been activated. These pin status values are tester output available to other functions for processing.



NODE: A31	**TITLE:** Process Test Data	**NUMBER:**

Figure A-14: Process Test Data

NODE A31: Process Test Data

Abstract

This node converts the test data from IC pin-number domain to physical location of ICUT pins on tester domain; changes the test data format to "SIEVE" format (a specific data format particularly required for simulation of Stanford IC tester). This node has been decomposed into four major functions Segregate Reference Data, Correlate Tester pins & Manual data, Correlate Tester pins & File Data and Convert Test data into SIEVE Format.

Process Test Data

This function has been sub-divided into four major sub-functions as shown in Fig. A-14. These sub-functions are:

(i) Segregate Reference Data: This function reads in data from ESIM (test data) file, separates out simulation data and expected output (reference data) for a particular input.

(ii) Correlate Tester Pins & Manual Data: This function sets correspondence between the test data received from the key-board in "Manual" mode of operation and location of ICUT pins on the IC tester.

(iii) Correlate Tester Pins & File Data: This function sets correspondence between the test data read from ESIM file in "Auto" mode of operation and location of ICUT pins on the IC tester.

(iv) Convert Test Data into SIEVE Format: This function changes the format of test data into "SIEVE" format. SIEVE is a special data format required for input to Stanford IC tester. An explanation of this data format is attached as Appendix "E".

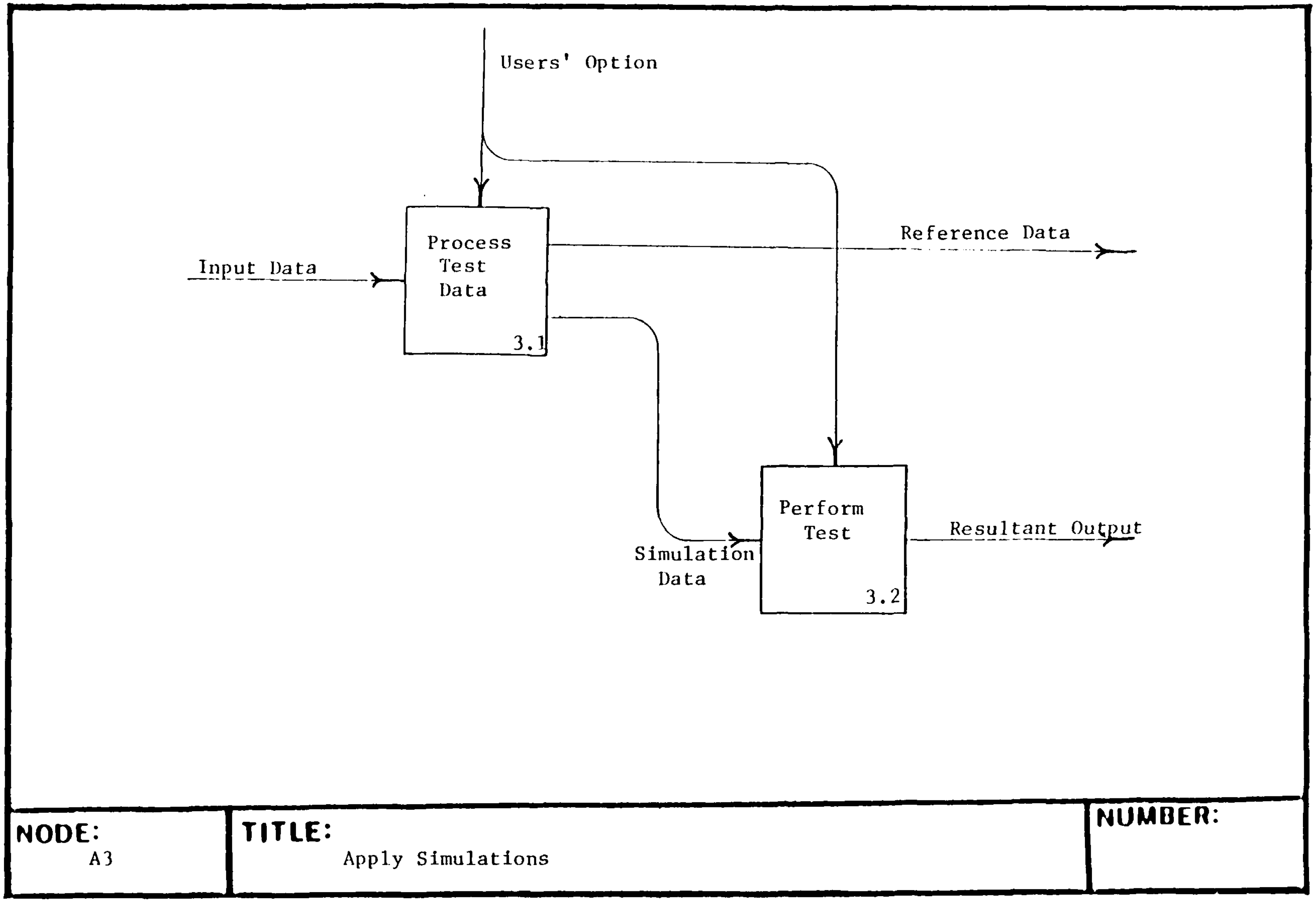


Figure A-13: Apply Simulations

NODE A3: Apply Simulations

Abstract

This node sets correspondence between physical location of ICUT pins on IC tester and test vectors in ICUT pin numbers' domain. It changes the test vector into SIEVE format, initializes IC tester and the ICUT. This node applies test vectors to simulate the ICUT, and samples the tester output for evaluation by other nodes.

Apply Simulations

This node is decomposed into two major functions as shown in Fig. A-13. These functions are:

(i) Process Test Data: This function receives test vectors from test data file in "Auto" mode of operation and from keyboard in manual mode of operation. It sets correspondence between ICUT pin numbers to be simulated by a test vector and their physical location on IC tester by consulting reference tables. This function also converts the test vectors into SIEVE format, which is a special data format required for simulation of Stanford IC Tester. An explanation of "SIEVE" format is attached as Appendix "E".

(ii) Perform Test: This function initializes the IC tester by applying voltages to significant pins to force out any ambigious voltage state from tester pins. It initializes the ICUT by applying initialization vectors in sequence to bring ICUT output pins to a steady state condition before proceeding with the actual test. This function also applies test vectors to IC tester to simulate ICUT and samples the ICUT for resultant output.

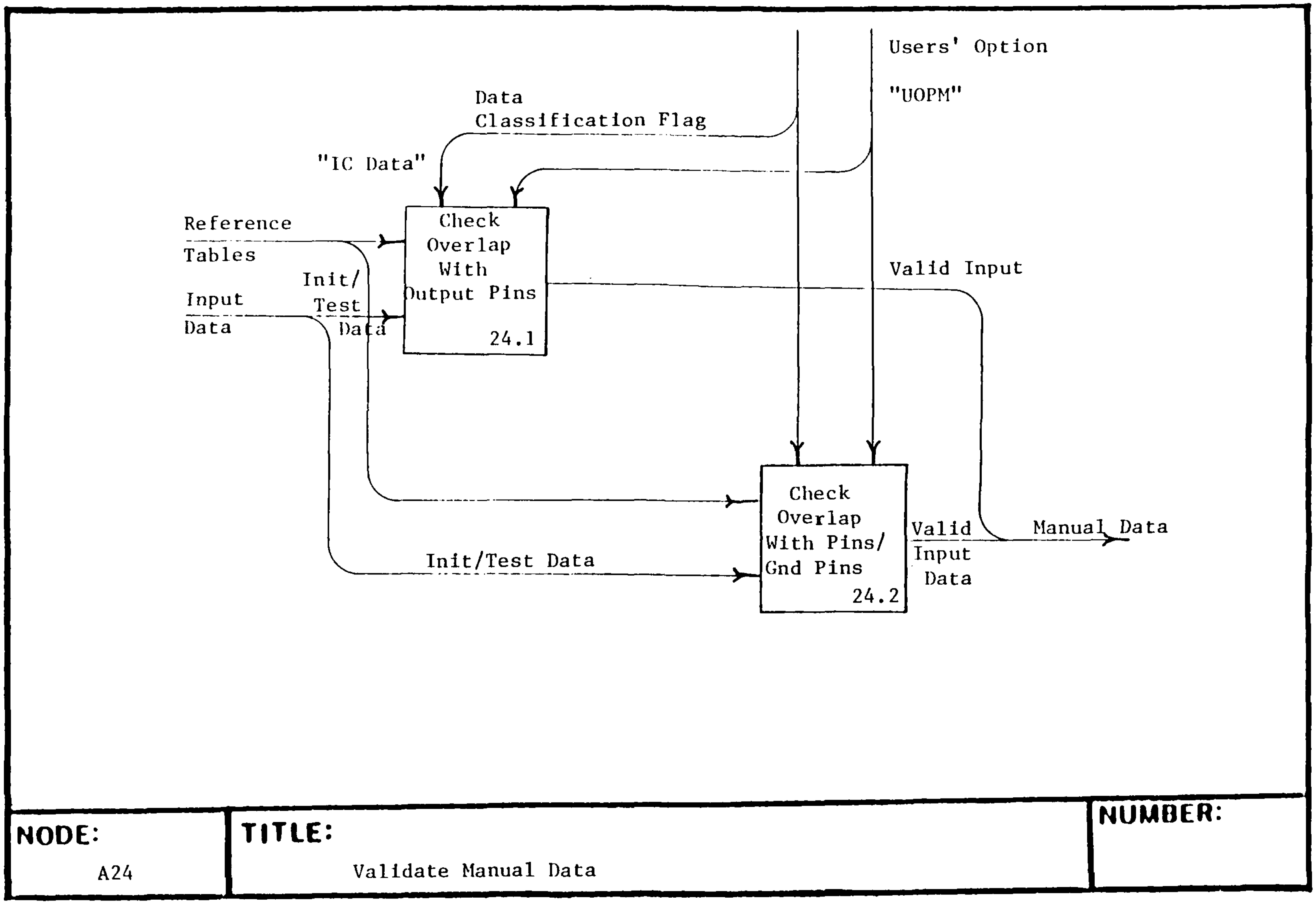


Figure A-12: Validate Manual Data

Node A24: Validate Manual Data

Abstract

This node is activated only in manual mode of operation. It checks for any overlap of input test data over designated output/power/ground pins.

Validate Manual Data

This node is decomposed into two functions as shown in Fig. A-12. These functions are:

(i) Check Overlap with Output Pins: This function carries out bitwise comparison of test vector to check if any pin designated as output pin is not simulated. It generates an error on detecting such an overlap and prompts user to modify his input data. This function is activated only in "Manual" mode of operation.

(ii) Check Overlap with Power/Ground Pins: This function carries out bitwise comparison of test vector to check if any pin designated as power or ground pin is not simulated. It generates an error on detecting such an overlap and prompts user to modify his input data. This function is activated only in "Manual" mode of operation.

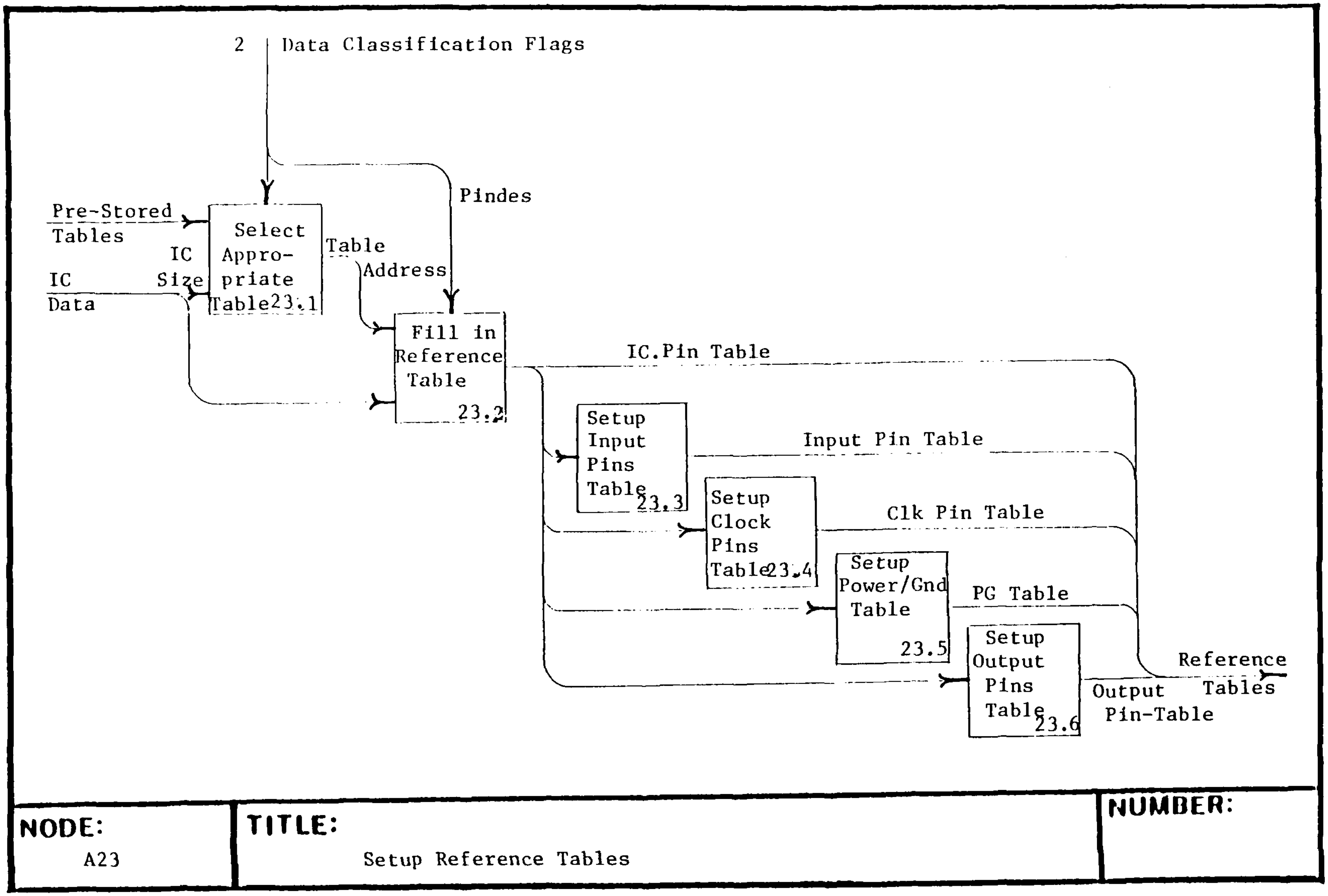

2
Data Classification Flags
Pre-Stored Tables
IC Size
IC Data
Select Appro-priate Table 23.1
Table Address
Pindes
Fill in Reference Table 23.2
IC.Pin Table
Setup Input Pins Table 23.3
Input Pin Table
Setup Clock Pins Table 23.4
Clk Pin Table
Setup Power/Gnd Table 23.5
PG Table
Setup Output Pins Table 23.6
Output Pin-Table
Reference Tables
NODE:
A23
TITLE:
Setup Reference Tables
NUMBER:

Figure A-11: Setup Reference Tables

NODE A23: Setup Reference Tables

Abstract

This node selects one of the prestored tables depending on physical characteristics of an ICUT. These reference tables contain information to set one to one correspondence between IC pins and IC tester pins for an IC of particular physical characteristics; it then prompts the user to provide information about input/output and other significant pins of the ICUT. This node also stores this data to correlate IC pin numbers in a test vector to their physical location on IC Tester for effecting a test simulation.

Setup Reference Tables

This node is decomposed into six functions as shown in Fig. A-11. These functions are:

(i) Select Appropriate Table: This function on receiving an input from keyboard, in response to a menu which prompts the user to select one of the five IC characteristics befitting the ICUT, makes the address of corresponding table available to the program. Five different tables for ICs with following characteristics have been prestored in the memory: 14, 16, 20, 24, 40pin dual-in-line packages. These tables correlate the IC pin numbers to IC tester pins when ICUT is positioned on the IC tester in a manner that pin1 of ICUT coincides with the top-left pin of IC tester.

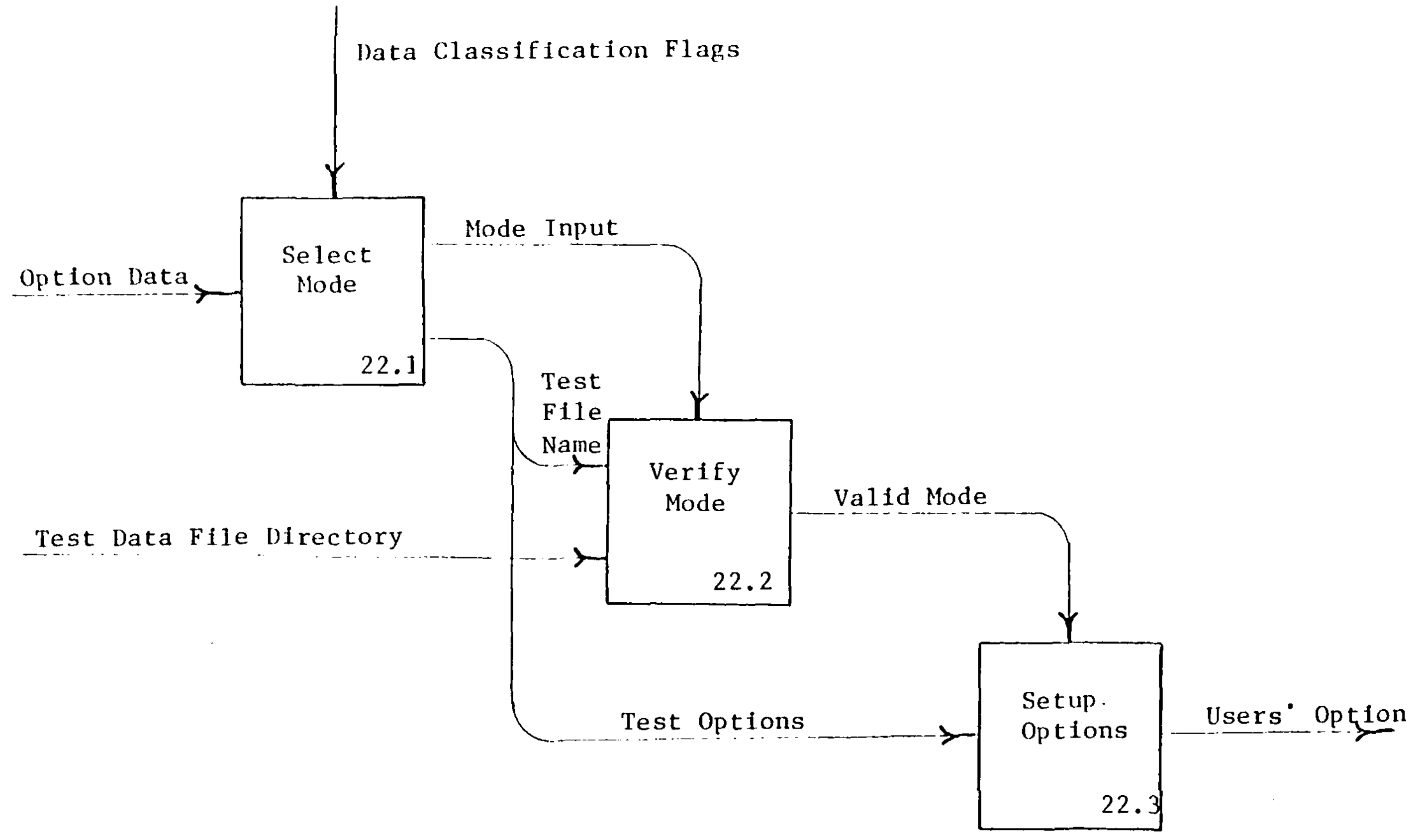
(ii) Fill in Reference Table: This function prompts the user to fill in complete reference table by inputting pin designation and class of each pin. (e.g., to provide information like pin1 = k-input "I", pin4 = phi2 "C", pin 14 = Vcc "P" and pin 7 = Gnd "G", etc.)

(iii) Setup Input Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "input" pins.

(iv) Setup Clock Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "output" pins.

(v) Setup Power/Ground Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "power/ground" pins.

(vi) Setup Output Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "output" pins.



NODE: A22

TITLE: Select Operating Options

NUMBER:

Figure A-10: Select Operating Options

NODE A22: Select Operating Option

Abstract

This node, receives option data from keyboard in response to different options made available to user in various menus, verifies their correctness and sets option flags to this effect.

Select Operating Option

This node is decomposed into three major functions as shown in Fig. A-10. These functions are:

(i) Select Mode: This function, in response to system prompt, receives IC nomenclature and preferred mode of operation data from keyboard. In addition, this function asks the user to input name of test file for "Auto" mode of operation.

(ii) Verify Mode: This function checks the IC nomenclature and searches the test data files' directory with test file name. It generates an error for non-availability of test file, informs the users to this effect and asks for another selection.

(iii) Setup Options: This function sets various option flags to be true/false for particular selection after user has opted for one of the operating choices offered to him in system operating menues.

All user options and corresponding flags are listed below:

UOPM	MANUAL mode. (abbreviated as MAN)
UOPM10	MAN with single step execution and terminal output only.
UOPM11	MAN with single step execution and terminal & file output.
UOPM20	MAN with multi-step execution & terminal output only.
UOPM21	MAN with multi-step execution and terminal & file output.
UOPA	AUTO mode
UOPA10	AUTO with "stop execution at first test failure" and terminal output only.
UOPA11	AUTO with "stop execution at first test failure" and terminal & file output.
UOPA21	AUTO with "stop execution after X-instruction" and terminal output only.
OUPA22	AUTO with "stop execution after X-instructions" and terminal & file output.
UOPA31	AUTO with "run whole test program printing all test failures" on terminal.
UOPA32	AUTO with "run whole test program printing all test failures" on terminal & storage file.

NODE A41: Analyze Results

Abstract

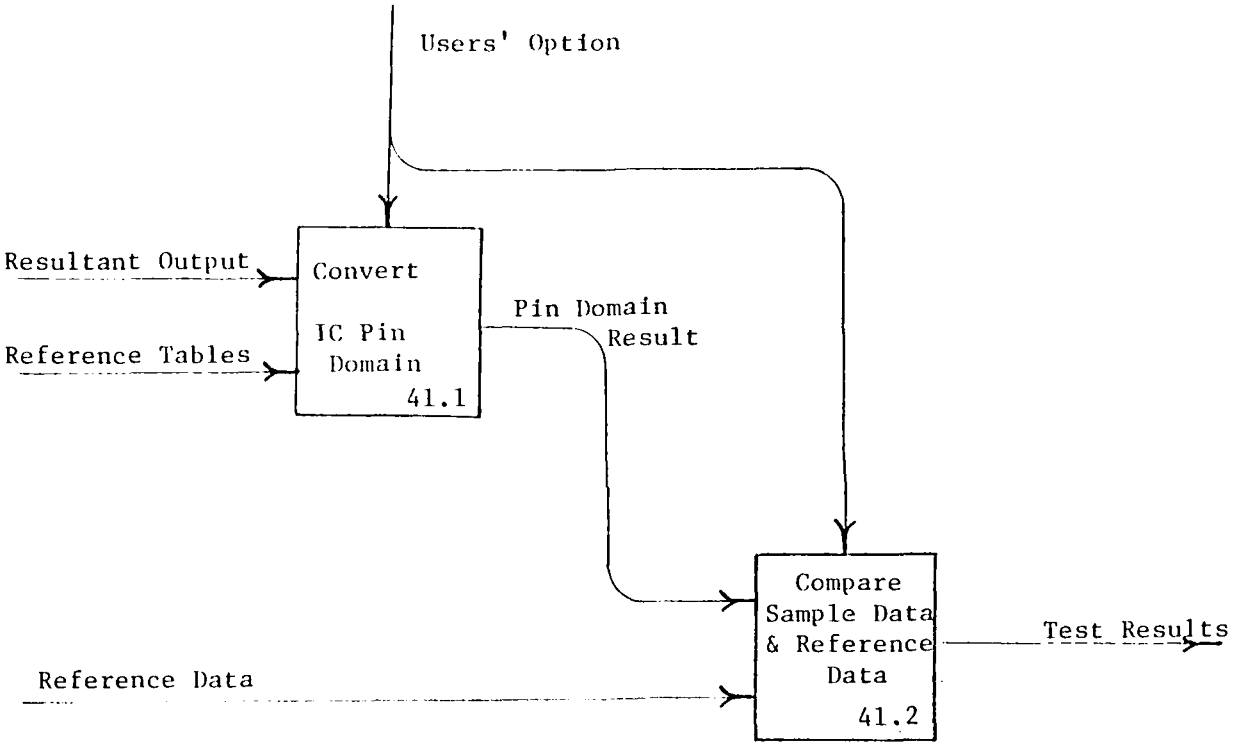
This node receives the resultant output of IC-tester, converts the output into ICUT pin domain from IC tester location reference by consulting reference tables. It also generates an error for any mismatch between the received output and expected output (reference data).

Analyze Results

This node is decomposed into two major functions as shown in Fig. A-17. These functions are:

(i) Convert Sample Data into IC Pin Domain: This function converts the sampled output of IC tester into ICUT pin numbers domain for comparison with the reference data. This conversion is achieved by consulting reference tables.

(ii) Compare Sample Data and Reference Data: This function compares output data and reference data bit by bit to single out any non-conformity between the two values. It generates a GO/NOGO message for successful/unsuccessful completion of a test and points out particular ICUT pins where output value does not match the reference value in case of a test failure.



NODE: A41

TITLE: Analyze Results

NUMBER:

Figure A-17: Analyze Results

NODE A42: Handle Results

Abstract

This node receives the test results, generates a test report depending on user's operating options, guides the resultant test report to display terminɑı and/or to a file for storage.

Handle Results

This node is decomposed into four functions as shown in Fig. A-18. These functions are:

(i) Formulate Test Report: This function generates test report from the test results and pre-stored messages, depending on successful/unsuccessful termination of a test.

(ii) Generate Storage Buffer: This function, allocates a buffer and stores all test reports generated during a test. This function is activated only if opted by user.

(iii) Copy Buffer to Disk: This function, not elaborated any further as it has been implemented by a library routine, copies the buffer containing test reports to a given file on disk. This function is also activated if opted.

(iv) Display Test Report: This function, depending on the user's operating option, displays all test reports on the terminal. This function has also been implemented by a library routine.

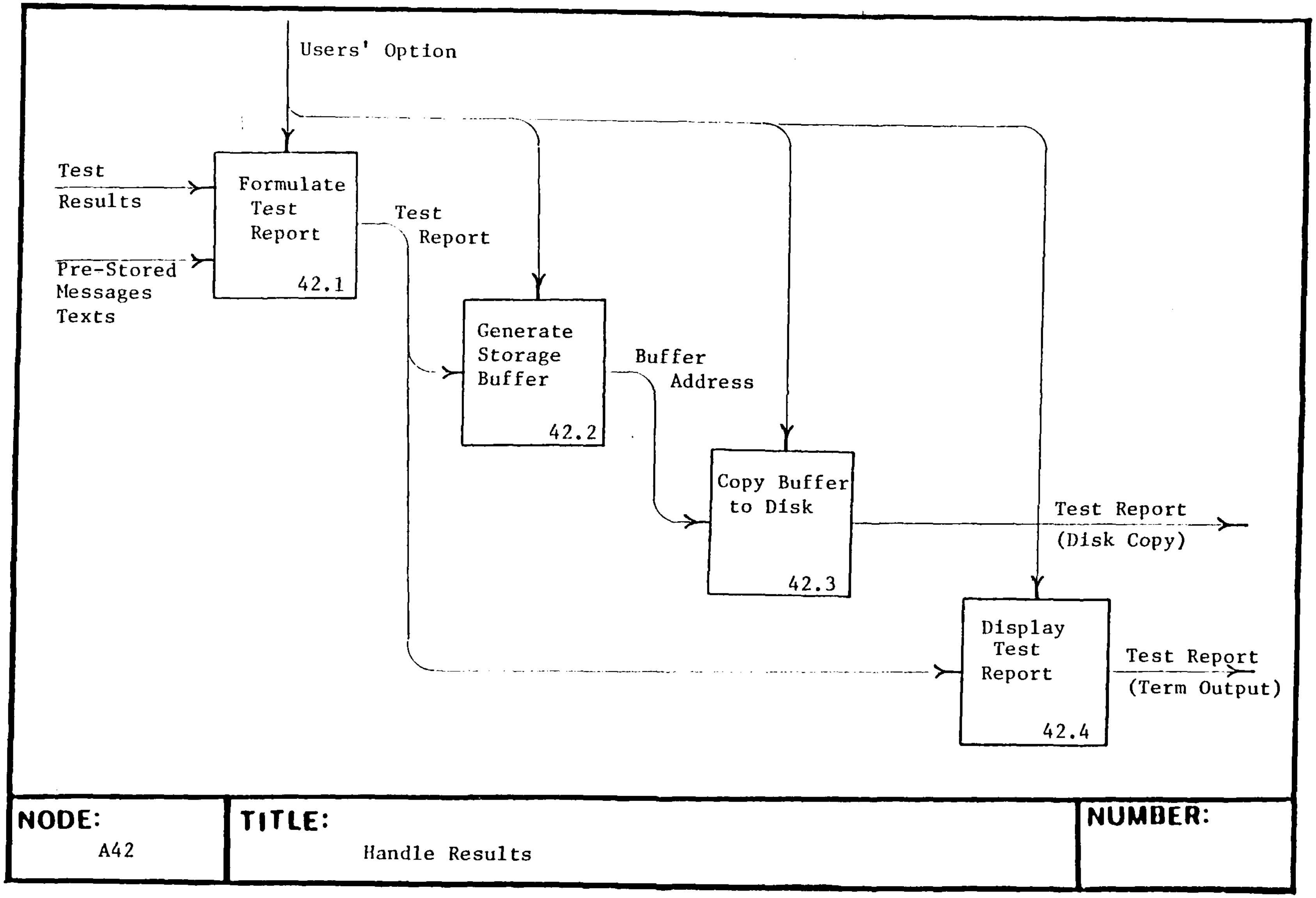


Figure A-18: Handle Results

Appendix "B"

Node List, Node Diagram and Node Descriptions

For Detailed Design of "Extract Test Data" Function

Extract Test Data

- Node A1: Extract Test Data
  - Node A11: Read ESIM File
    - Node A111: Get Text-Line
    - Node A112: Classify Text-Line
      - Node A1121: Read First Character
      - Node A1122: Check if Alpha Numeric
      - Node A1123: Check if GT
  - Node A12: Tabulate Command Data
    - Node A121: Check for Specific Command
    - Node A122: Tabulate Test Pins
      - Node A1221: Create Array of Monitored Pins
      - Node A1222: Create Array of Clock Pins
      - Node A1223: Create Array of Input Pins
      - Node A1224: Generate Array of Output Pins
        - Node A12241: Mark Inpins
        - Node A12242: Mark Clk Pins
        - Node A12243: Create Outpin Array
    - Node A123: Handle Remaining Commands
      - Node A1231: Validate Command Line

      - Node A12311: Create Array of Effected Pins
      - Node A12313: Check if Element of Outpin Array
    - Node A1232: Change Effected Pins' Status
- Node A13: Restiucture Test Data
  - Node A131: Store Node Data
    - Node A1311: Segregate Data & Node Name
    - Node A1312: Locate Test Node in Monpin Array
    - Node A1313: Add Data in Associated Node Buffer
      - Node A13131: Gauge Data in Buffer
      - Node A13132: Gauge Incoming Data
      - Node A13133: Check Overflow
      - Node A13134: Add New-Data to Pre-Data
  - Node A132: Change Data Structure
    - Node A1321: Segregate Input & Output Nodes
    - Node A1322: Convert Node Data Into Vectors
- Node A14: Change Data Format

NODE A1: Extract Test Data

Abstract

This node scans the ESIM file to separate out pertinent test data. It classifies the extracted data into test input data and reference output data. This function restructures this data available in node configuration into vector form. It also changes the memory storage pattern of restructured test data from VAX computer system to RT-11 Operating System data format.

Extract Test Data

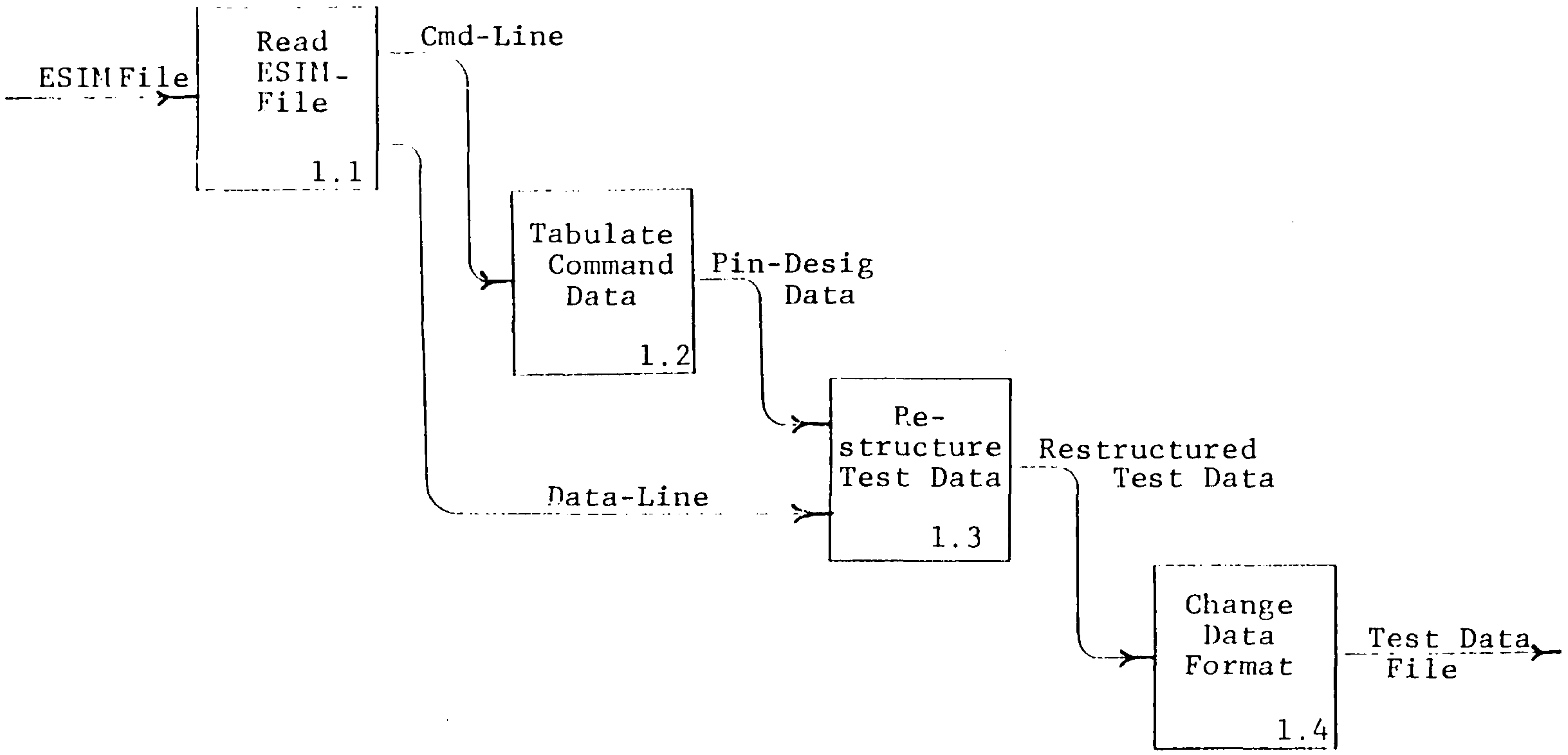
This node is decomposed into four major functions as shown in Figure B-1. These functions are:

(i) Read ESIM File: This function reads in the given ESIM File line by line and classifies each incoming line into command-line or data-line. A brief explanation of all commands encountered in ESIM Files is attached as Appendix "F".

(ii) Tabulate Command Data: This function interprets the commands in a given command-line to setup different arrays to contain names of all monitored pins, input pins and clock pins. This function also generates an array of output pins from the already available pin lists.

(iii) Restructure Test Data: This function interprets the data line and stores all the test data pertaining to a node into its respective buffer. This function does some housekeeping by converting the node data into vector form and storing it in an external file to empty buffer for the following node data. This is done when a change in pin-designation is made or data buffer overflows due to incoming data.

(iv) Change Data Format: This function changes the memory storage pattern of restructured test data from VAX computer system into RT-11 data format. This function is implemented by a system library routine and therefore, it is not being elaborated any further.



NODE: A1

TITLE: Extract Test Data

NUMBER:

Figure B-1: Extract Test Data

NODE All: Read ESIM File

Abstract

This node reads in given ESIM file, line by line until the end of file is reached and classifies each incoming line to be either a command-line or a data-line.

Read ESIM File

This node is decomposed into two functions as shown in Figure B-2. These functions are:

(i) Get Text-Line: This function reads a given ESIM file line by line until end of file and is implemented by a system library routine (fgets). This function is not elaborated any further.

(ii) Classify Text-Line: This function, on receiving a text-line from ESIM file categorizes it to be either a command line if first character of the test-line is "alphabetic" or a data-line if first character of given text-line is ">". A brief explanation of terminology used in ESIM files is attached as Appendix "F".

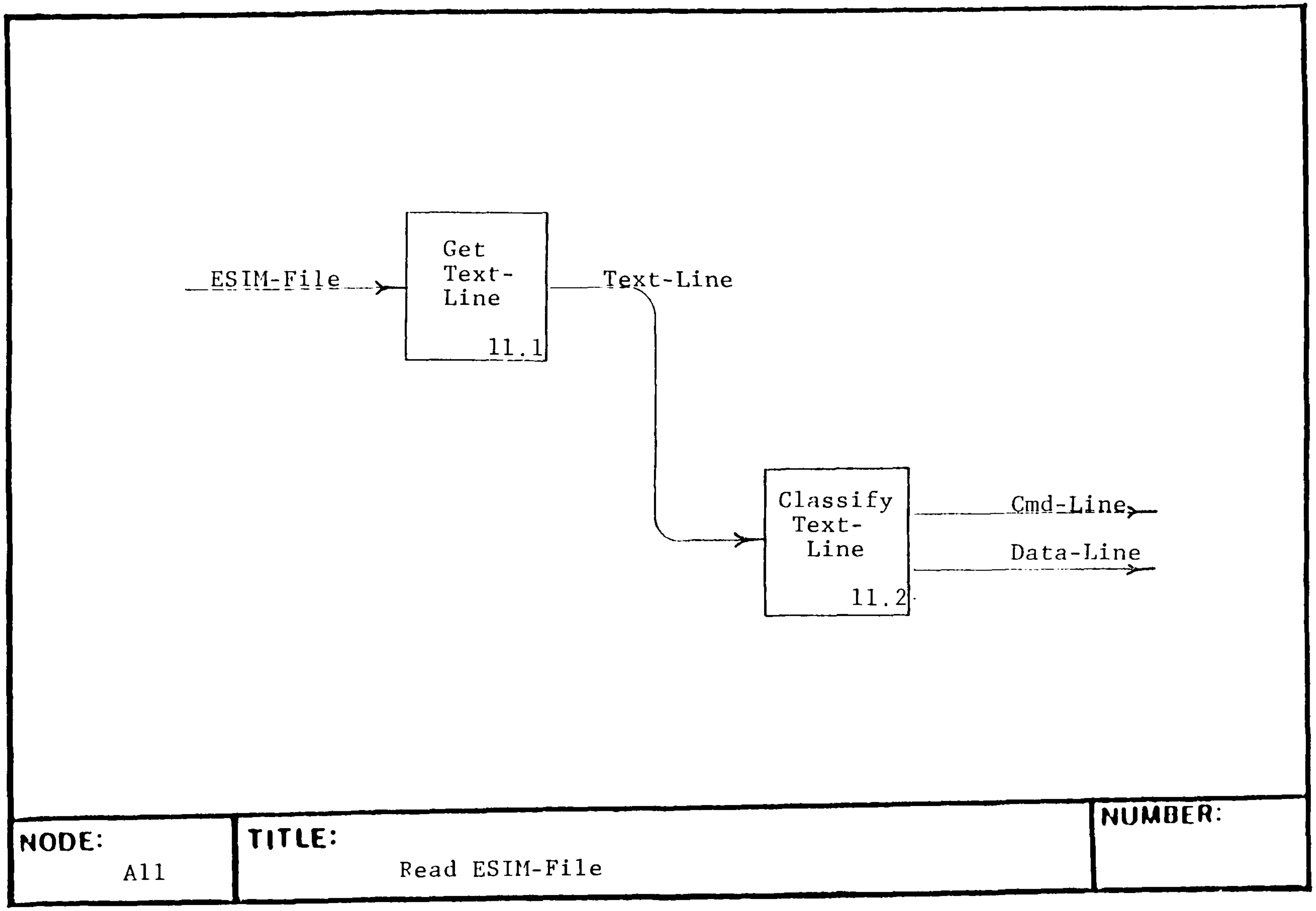


Figure B-2: Read ESIM-File

NODE A112: Classify Text-Line

Abstract

This node interprets the first character of the text-line and classifies the text line to be either a cmd-line or a data-line.

Classify Text-Line

This node is decomposed into three functions as shown in Figure B-3. These functions are:

(i) Read First Character: This function reads the first character of a given text line. This is implemented by considering text line to be a an array of characters and accessing the array element at subscript [0].

(ii) Check if Alphabetic: This function confirms if first character of the given text-line is alphabetic. This is implemented by a system library subs-routine (isalpha( )).

(iii) Check if GT: This function compares the given character with character ">" (greater than sign) to confirm if text-line is a data line. This is implemented by "C" language relation operator "==".

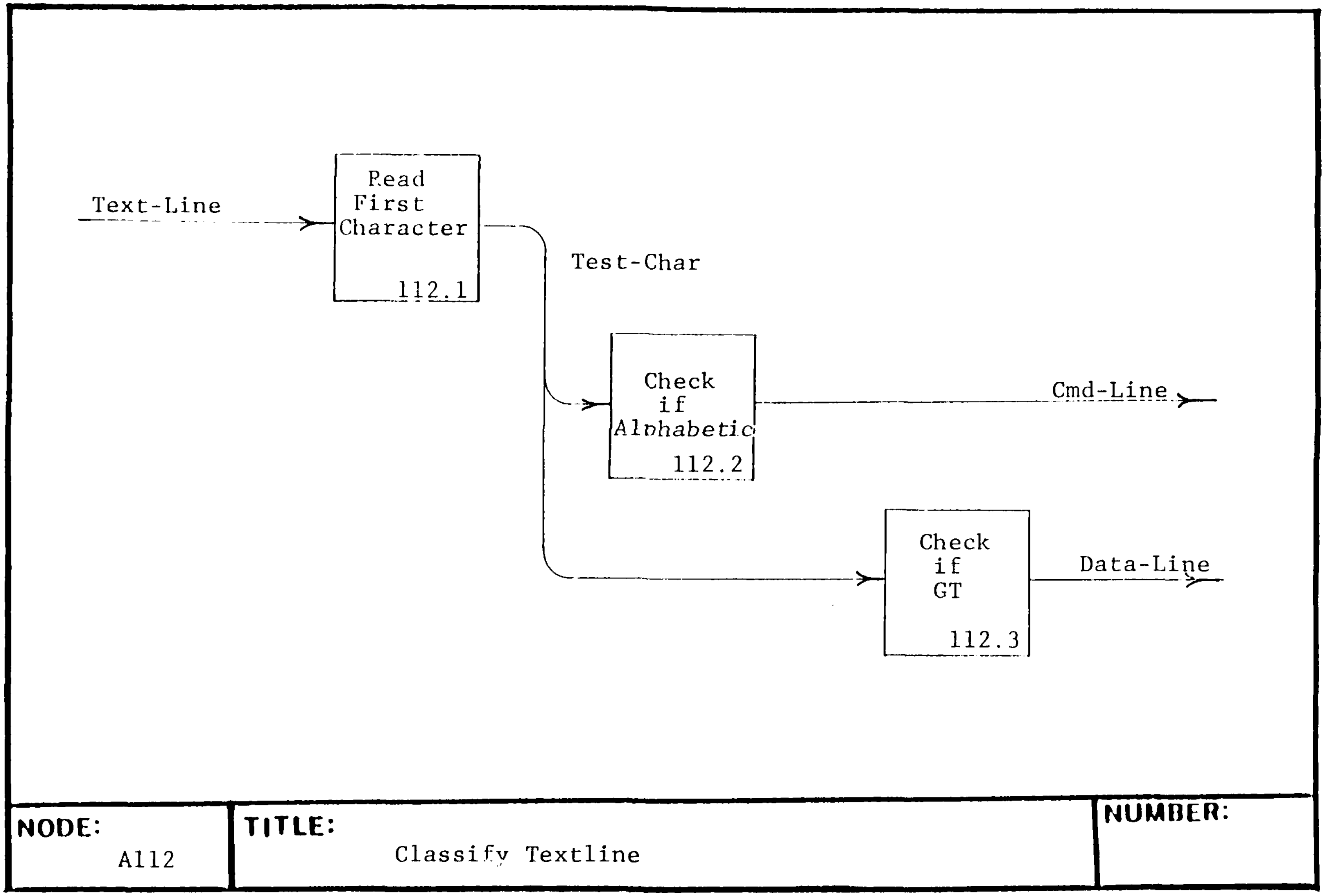


Figure B-3: Classify Textline

NODE A12: Tabulate Command Data

Abstract

This node interprets the command in a given command line to setup arrays of monitored pins, input pins and clock pins. This function generates an array of output pins from available pin-data and also changes the status of any designated pin on receiving specific instructions.

Tabulate Command Data

This node is decomposed into three major functions as shown in Figure B-4. These functions are:

(i) Check for Specific Command: This function singles out first character of a given command line and branches out to perform specific sub-function on its interpretation.

(ii) Tabulate Monitored Pins: This function generates arrays of monitored pins, input pins and clock pins. It establishes reference between all three arrays. This function also stores available initialization data and clock sequences and establishes correspondence between this data and related pins.

(iii) Handle Remaining Commands: This function, on interpreting the first character of command line to be "h or l", scans arrays of output pins and clock pins to check if any pin from these categories has not been forced to hold a status of "high/low" voltage level. This function changes the status of those pins which are outside above two categories of input and output pins.

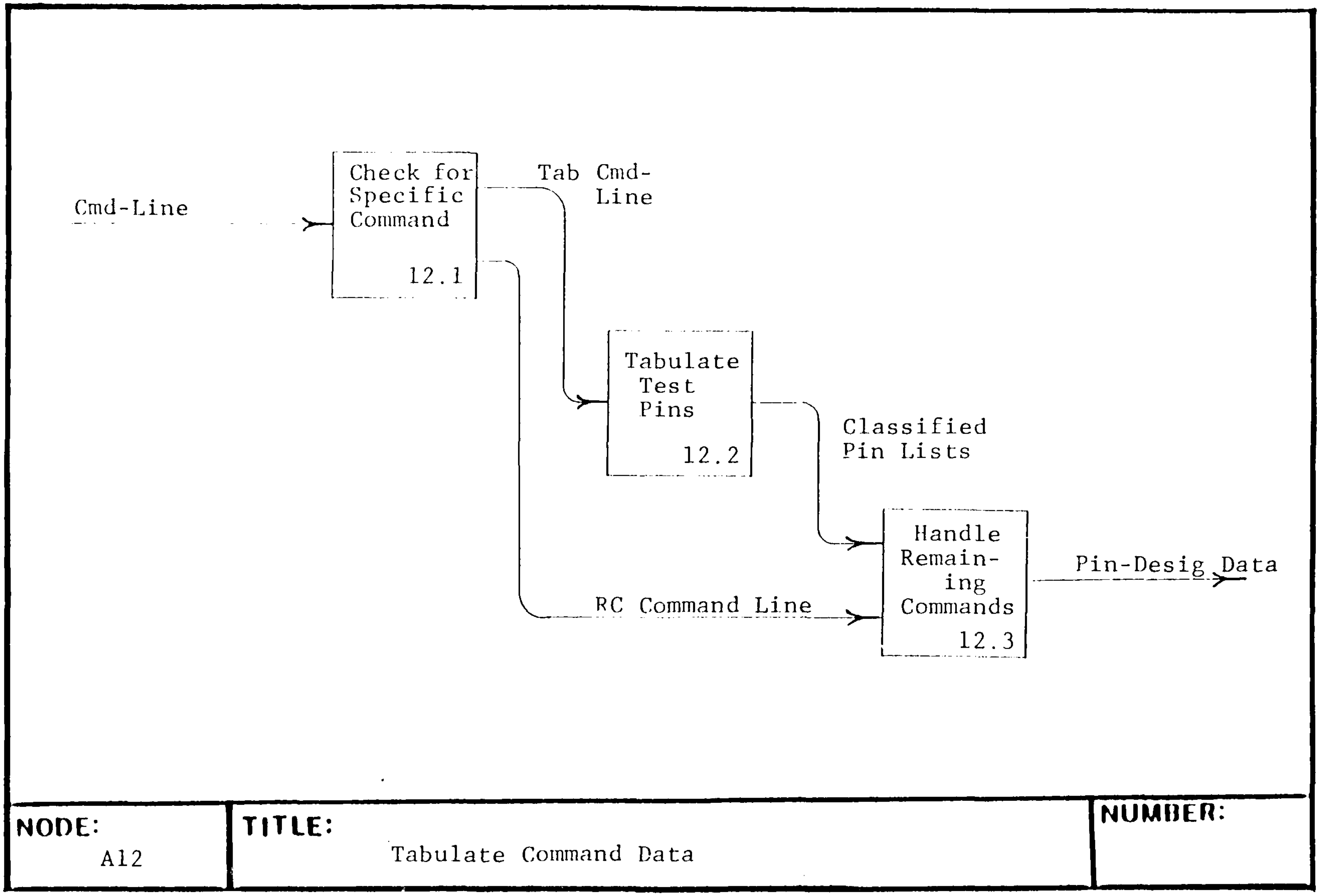
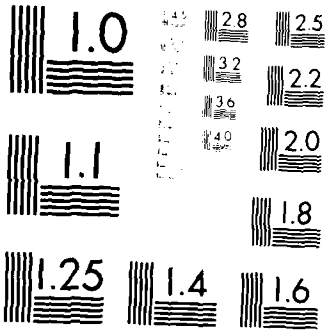


Figure B-4: Tabulate Command Data

AD-A151 961 SYSTEM DESIGN OF AUTOMATED VLSI (VERY LARGE SCALE 2/4
INTEGRATED) TEST STATIO..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. S TARIQ
UNCLASSIFIED DEC 84 AFIT/GE/EE/84D-27 F/G 9/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
1.1
2.0
1.8
1.25
1.4
1.6

NODE A121

Abstract

This node interprets first character of a given command line and branches out to perform specific sub-function.

Check for Specific Command

This function on detecting the first character of the command-line to be W, V, K, or I, branches out to tabulate the information in the given "command line". For other commands like "h, l, x, N", function "handle remaining commands" is activated.

This function is implemented by "C" language "switch statement". It will not be elaborated any further. (Figure B-5)

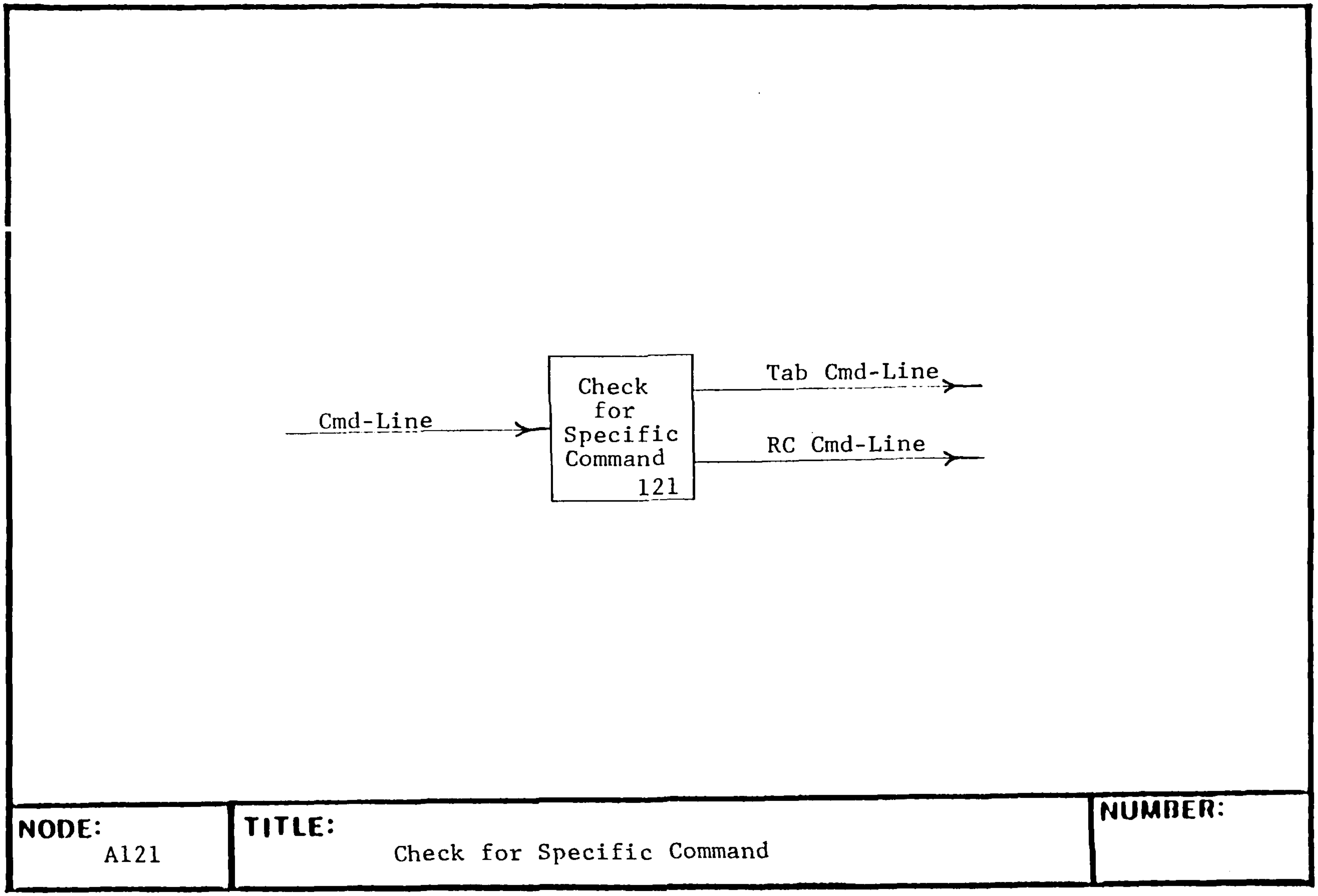


Figure B-5: Check for Specific Command

NODE A122: Tabulate Test Pins

Abstract

This node on receiving specific commands sets up arrays of monitored pins, input pins and clock pins. It establishes a cross reference between these array elements and between each array element and data related to it. This function also generates an array of output pins.

Tabulate Monitored Pins

This node is decomposed into four major functions as shown in Figure B-6. These functions are:

(i) Create Array of Monitored-Pins: This function, on interpreting first character of command line to be "w", sets up an array of all pin names included in the remaining command line.

(ii) Create Array of Input-Pins: This function, on interpreting first character of command line to by "V", sets up an array of input-pin and adds the pin name in this and all subsequent "input" command-lines to this array. It establishes a reference between each input pin and data related to it. This function also sets up cross reference between elements of this array and array of monitored-pin.

(iii) Create Array of Clock-Pins: This function, on interpreting first character of command line to be "K", sets up an array of all pin names included in the remaining command line. It establishes a reference between each clock pin and clocking sequence for this pin. This function also sets up cross reference between elements of this array and array of monitored-pins.

(iv) Generate Array of Output Pins: This function on interpreting the first character of command line to be "I", segregates all pins from array of monitored pins not marked as input or clock pins, and lists them into a separate array.

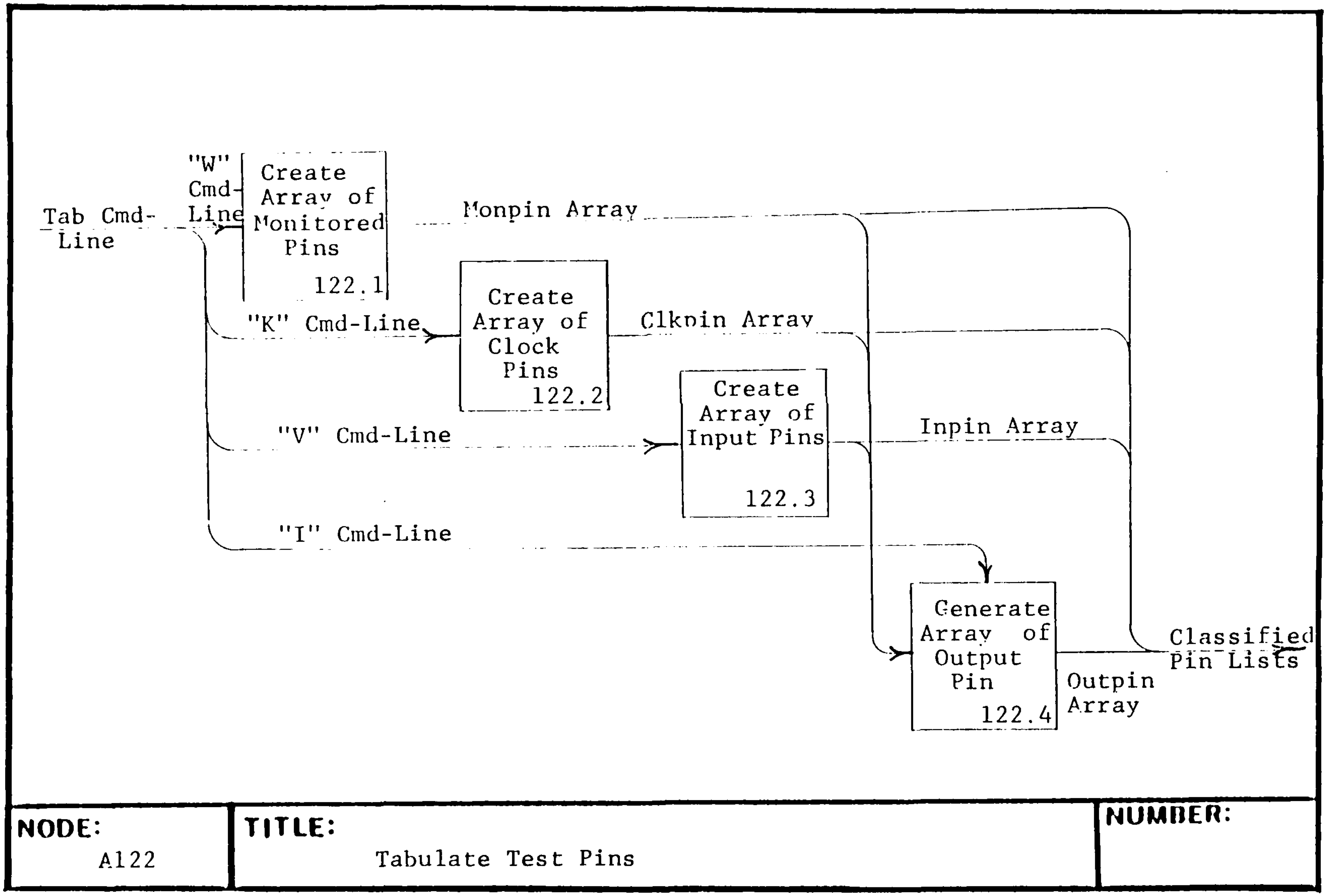


Figure B-6: Tabulate Test Pins

NODE A1224: Generate Array of Output Pins

Abstract

This node classifies the pins in array of monitored pins for being clock pins or input pins, and creates a separate array of output pins from unspecified pins in array of monitored pins.

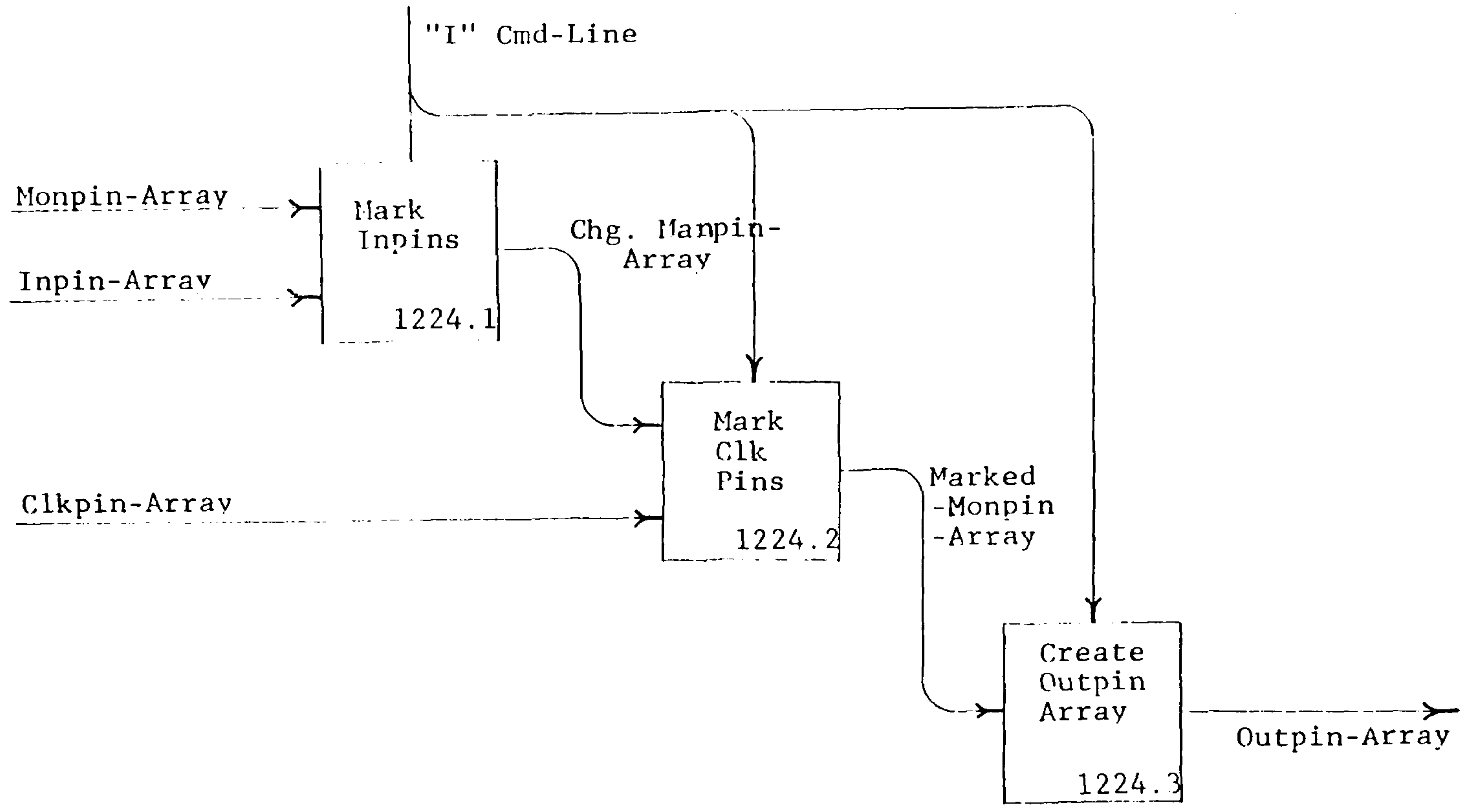
Generate Array of Output Pins

This node is decomposed into three major functions as shown in Figure B-7. These functions are:

(i) Mark Inpins: This function classifies the pins included in array of monitored pins to be input pins if these are also elements of inpin-array.

(ii) Mark Clkpins: This function classifies the pins included in array of monitored pins to be clock pins if these are also elements of clkpin-array.

(iii) Create Outpin-Array: This function scans the array of monitored pins and creates another array consisting of pins not marked as input/clock pins.



NODE: A1224	**TITLE:** Generate Array of Output Pins	**NUMBER:**

Figure B-7: Generate Array of Output Pins

NODE A123: Handle Remaining Commands

Abstract

This node validates the commands which effect the status of a designated pin. It generates an error for any invalid command or otherwise changes the status of the effected pin.

Handle Remaining Commands

This node is decomposed into two functions as shown in Figure B-8. These functions are:

(i) Validate Command Line: This function, on interpreting the first character of a given command line to be "h or l", scans the arrays of output pins and clock pins to check if status of any pin from these categories has been effected. It marks the command to be "valid" if no pin from the above two classes is effected by the command.

(ii) Change Effected Pins' Status: This function changes the status of effected pin, which belongs to the categories of either input pins or unmarked pins, and holds it until any new command is received to change it status.

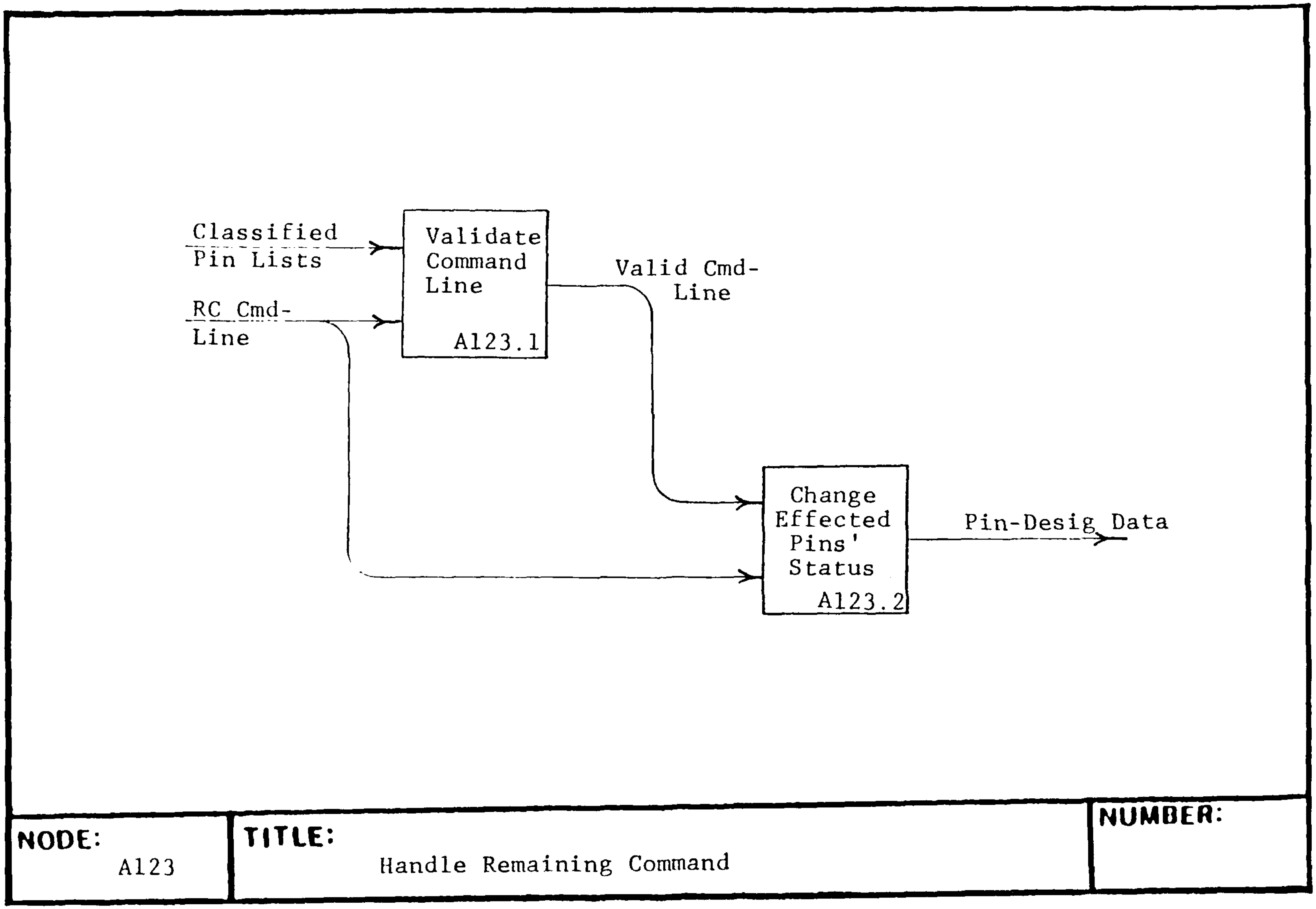


Figure B-8: Handle Remaining Command

NODE A1231: Validate Command Line

Abstract

This node ensures that any command line may not force an output pin to a "high/low" voltage status permanently.

Validate Command Line

This node is decomposed into two functions as shown in Figure B-9. These functions are:

(i) Create Array of Effected Pins: This function, on interpreting tne first character of the RC Command-Line to be "h" or "l" sets up an array of all remaining pins included in the command line.

(ii) Check If Element of Outpin Array: This function gets each element of newly created "effected pin array" and scan outpin array for its match. It declares RC Cmd-Line to be a valid Cmd-Line if no match is found.

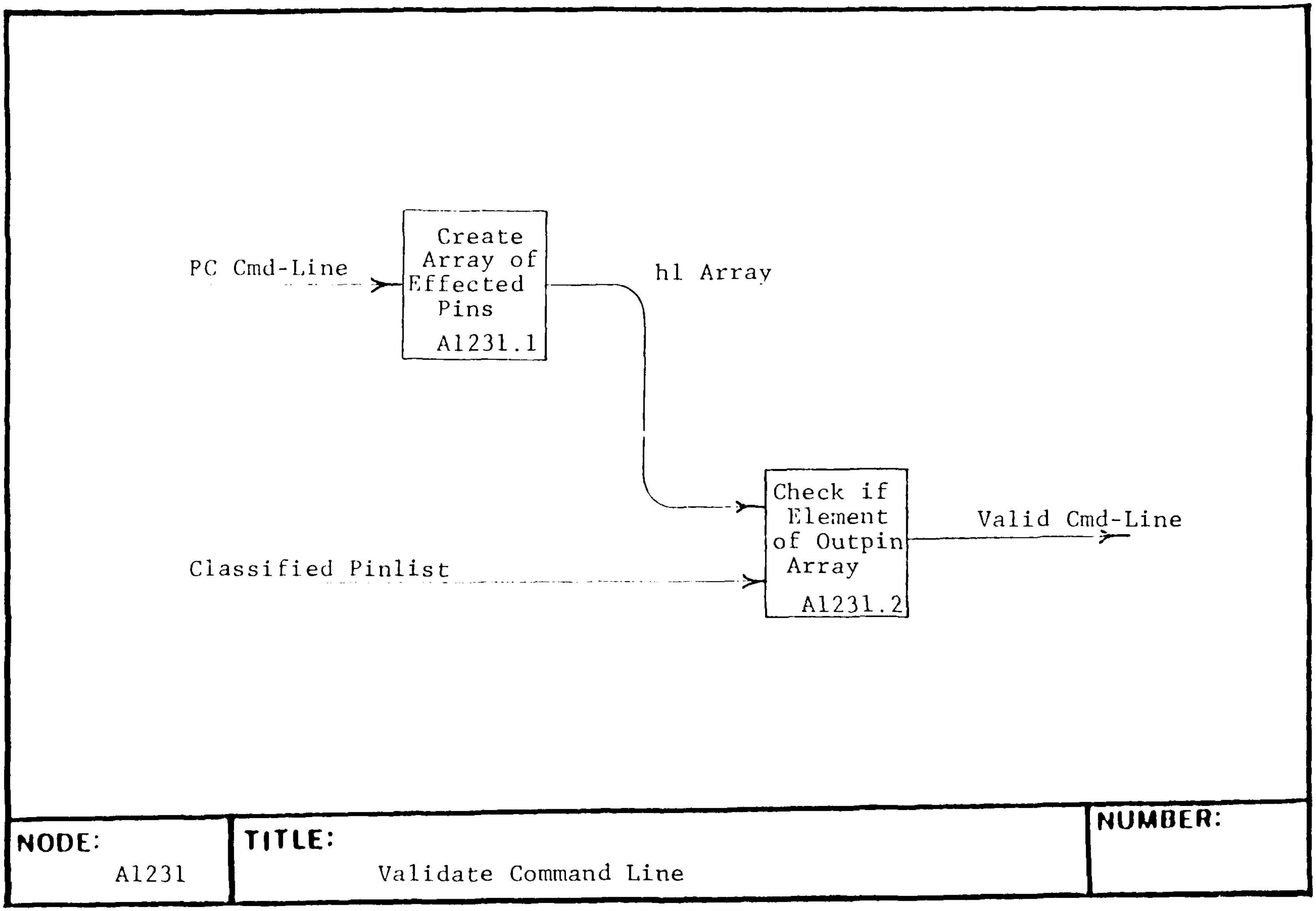


Figure B-9: Validate Command Line

NODE A13: Restructure Test Data

Abstract

This node interprets the data line and stores all the test data pertaining to a node into its respective buffer. This node does some housekeeping by converting the node data into vector form and storing it in an external file to empty buffer for the following node data. This is done when a change in pin-designation is made or data buffer overflows due to incoming data.

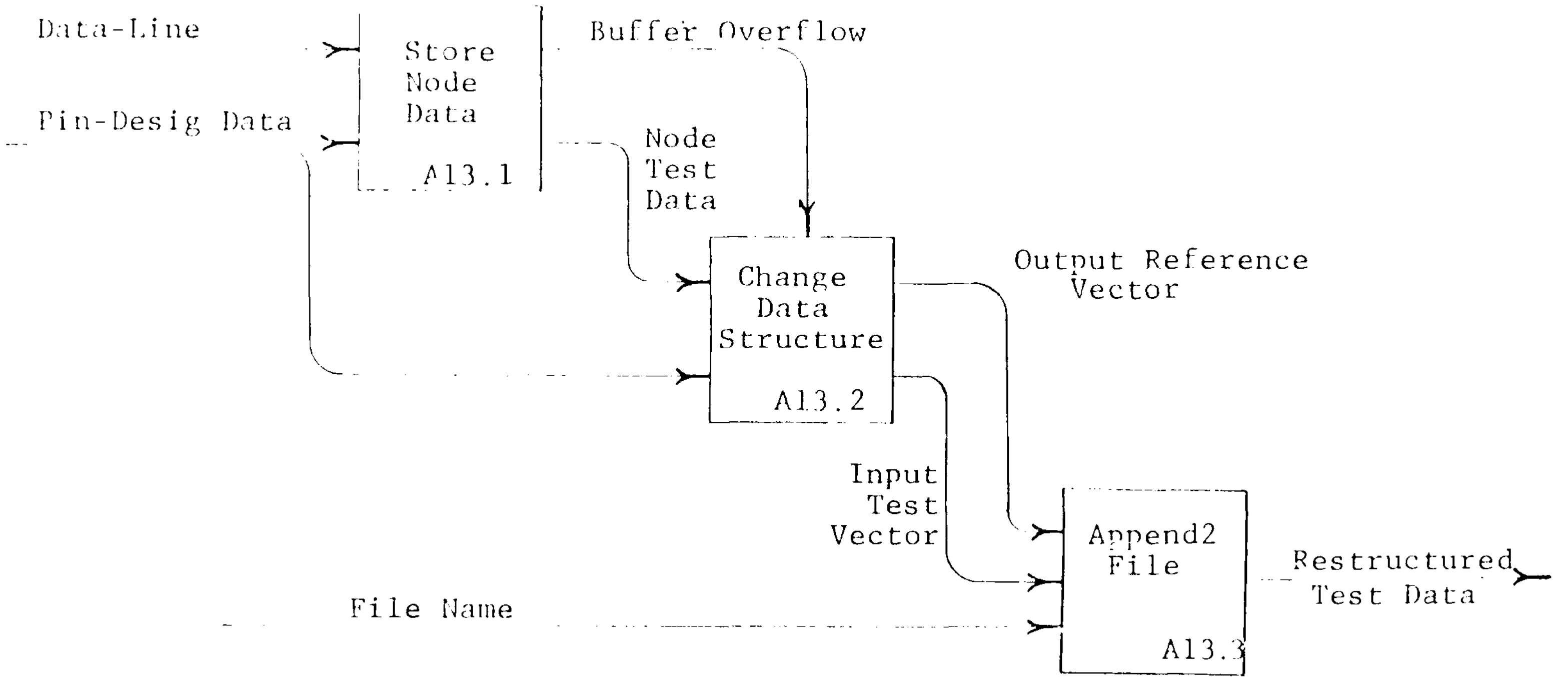
Restructure Test Data

This node is decomposed into four major functions as shown in Figure B-10. These functions are:

(i) Store Node Data: This function on receiving a data line, interprets the specific node for which data line is intended and add the data to a buffer attached to that node. It generates an overflow error if the incoming data exceeds the capacity of respective buffer.

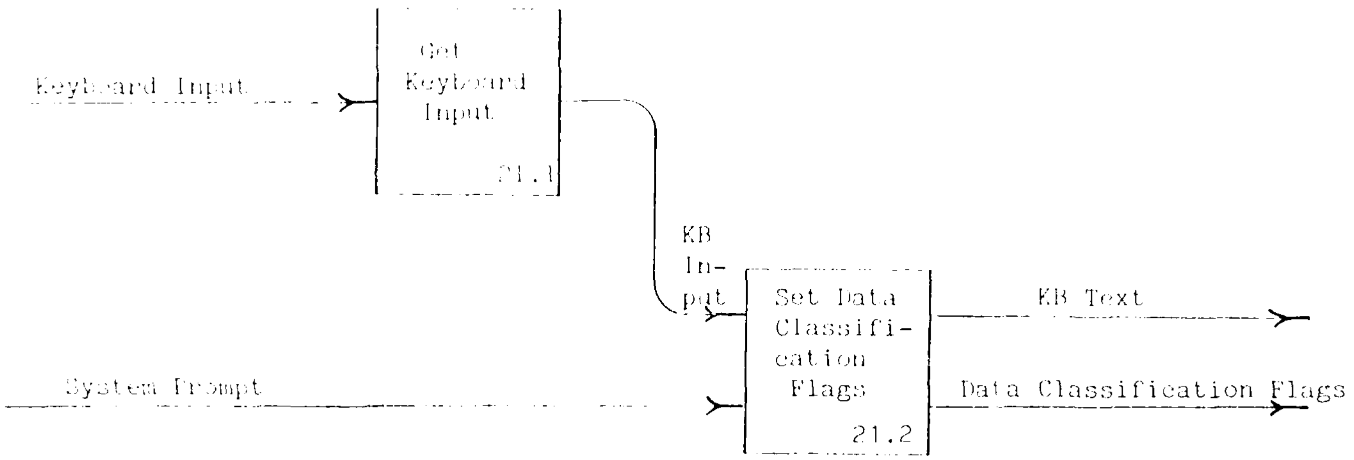
(ii) Change Data Structure: This function converts the available test data in node configuration into vector form. It changes the node data related to input pins into test vectors and node data related to output pins into reference vectors.

(iii) Append2-File: This function empties the buffer by writing all restructured data to an external file, whose name has been provided by the user.



NODE: A13	TITLE: Restructure Test Data	NUMBER:

Figure B-10: Restructure Test Data



NODE: A21	**TITLE:** Classify Input	**NUMBER:**

Figure C-2: Classify Input

Node A21: Clasify Input

Abstract

This node receives all command/data input from keyboard, checks its syntax and categories the input into three broad classes: IC Data, Option Data and Test Data.

Classify Input

This node is decomposed into two major functions as shown in Figure C-2. These functions are:

(i) Get Keyboard Input: This function scans input port and gets any character or character string input from keyboard in response to system prompt. This function is implemented then routines available in system library, and therefore, it will not be elaborated any further.

(ii) Set Data Classification Flags: This function sets one of three data classification flags from system prompt to classify the input in three broad categories of IC Data, Option Data and Test Data for further processing.

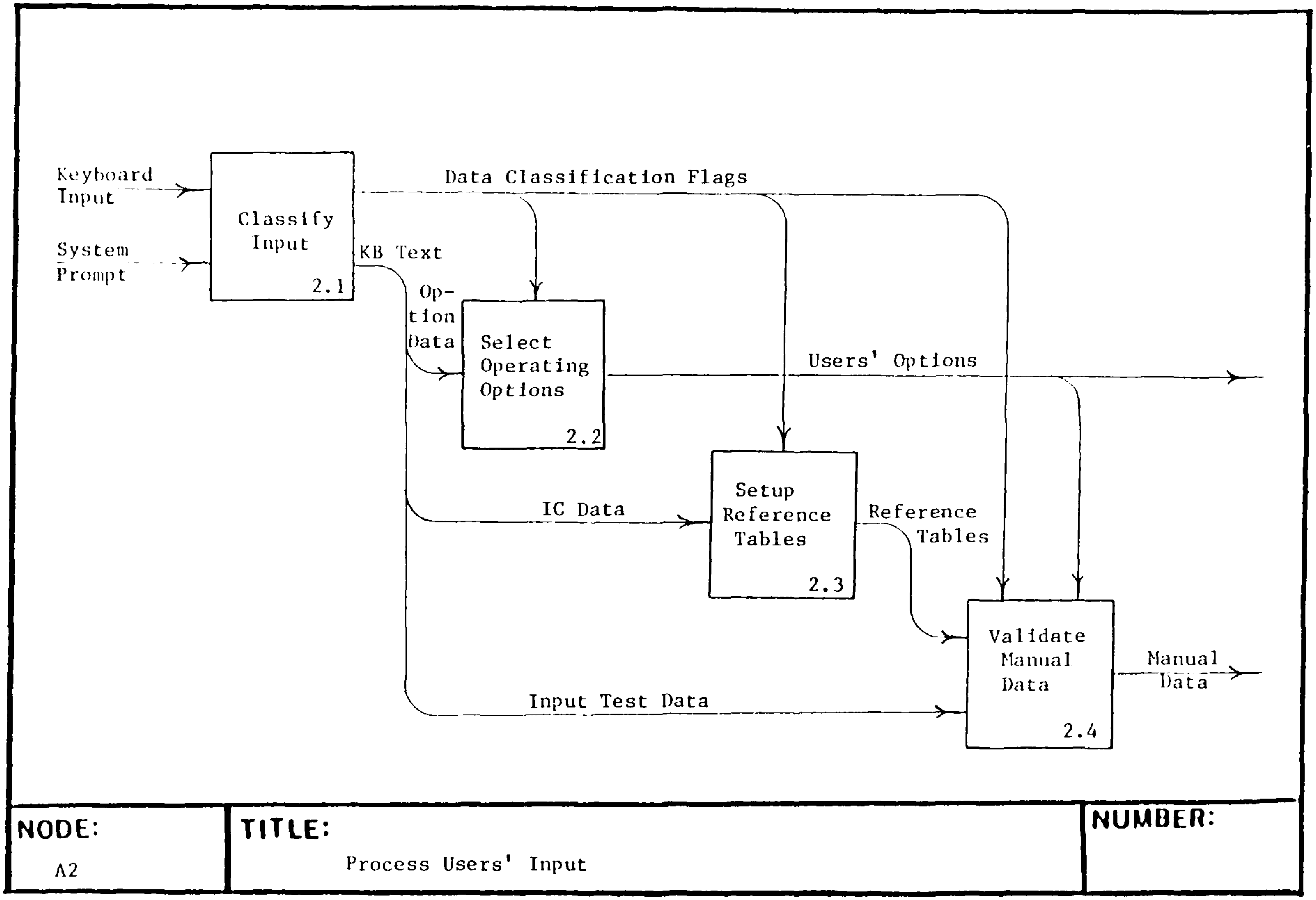


Figure C-1: Process Users' Input

Node A2: Process Users' Input

Abstract

This node receives all command/data input by the user in response to system prompts, checks its syntax and sets various flags for operating mode conditions, sets up reference tables to correlate IC pin numbers with their physical location on IC Tester and validates test data for "Manual" mode of operation.

Process Users' Input

This node is decomposed into four major functions as shown in Figure C-1. These functions are:

(i) Classify Input: This function receives all characters or character strings from keyboard as command or data input in response to system prompt and classifies all inputs into three broad categories, IC Data, Option Data, and Test Data. It also sets various flags under system prompt to aid in classification of incoming data.

(ii) Select Operating Options: This function allows the user to operate in any desired mode from a range of selections offered in system menues. This function sets different flags to run the program according to users' option.

(iii) Setup Reference Tables: This function selects one of the prestored tables depending on physical characteristics of ICUT. These reference tables contain information to set one to one correspondence between IC pins and IC tester pins for an IC of particular physical characteristics. It then prompts the user to provide information about input/output and other significant pins of ICUT. This function stores this data to correlate IC pin numbers in a test vector totheir physical location on IC Tester for effecting a test simulation.

(iv) Validate Manual Data: This function is activated only in manual mode of operation. It checks for any overlap of input test data over designated output/ppower/ground pins.

Node A23: Setup Reference Table

Node A231: Select Appropriate Table

Node A2311: Propmt User to Mark IC-Char

Node A2312: Get Ic Characteristics

Node A2313: Check Syntax of Selection

Node A2314: Locate Appropriate Table

Node A232: Fill In Reference TAble

Node A2321: Get Pin-Designation

Node A2322: Get Pin-Class

Node A2323: Store Data in Selected Table

Node A233: Setup Input Pins' Table

Node A2331: Locate Class "I" Pin

Node A2332: Setup Array of Input Pins

Node A234: Setup Clock Pins Table

Node A2341: Locate Class "K" Pin

Node A2342: Setup Array of Clkpins

Node A235: Setup Pwr/Gnd Table

Node A2351: Locate Class "P/G" Pin

Node A2352: Setup Array of "P/G" Pins

Node A236: Setup Output Pins Table

Node A2361: Locate Class "O" Pin

Node A2362: Setup Array of Output Pins

Node A24: Validate Manual Data

Node A241: Check Overlap with Output Pins

Node A242: Check Overlap with Pwr/Gnd Pins

Appendix "C"

Node List, Node Diagrams & Node Descriptions

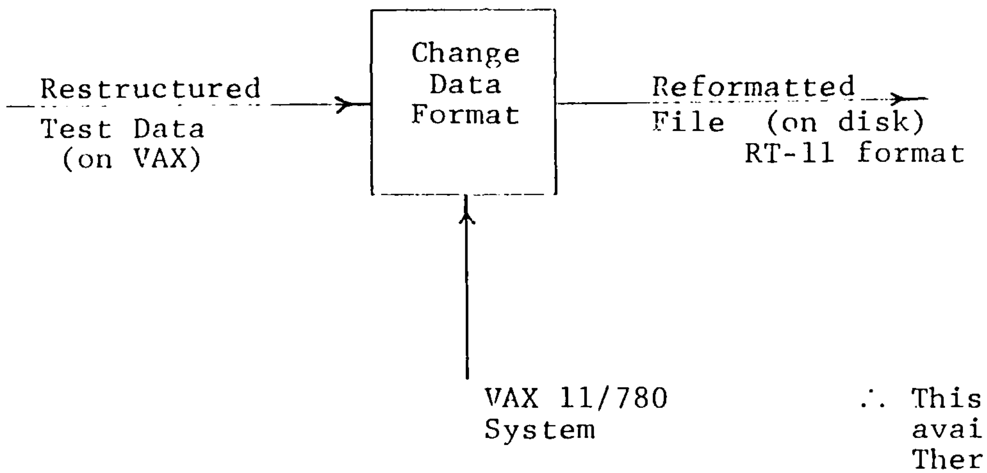
For Detailed Design of "Process Users' Input Function

Process Users' Input

Node List

Node A2: Process Users' Input

- Node A21: Classify Input
  - Node A211: Get Keyboard Input
  - Node A212: Set Data-Classification Flags
- Node A22: Select Operating Option
  - Node A221: SAelect Mode
    - Node A2211: Get Mode
    - Node A2212: Set Flag for Auto Mode
    - Node A2213: Prompt for Name of Test File
    - Node A2214: Store Text File Name
  - Node A222: Verify Mode
    - Node A2221: Check Availability of Test File
    - Node A2222: Confirm Mode Validity
  - Node A223: Setup Options
    - Node A2231: Prompt User to Select Option
    - Node A2232: Get Option Response
    - Node A2233: Check Option Syntax
    - Node A2234: Set Option Flag



∴ This program is already available on VAX system Therefore, it is not elaborated any further.

NODE: A14	**TITLE:** Change Data Format	**NUMBER:**

Figure B-14: Change Data Format

NODE A14: Change Data Format

Abstract

This node changes the memory storage pattern of restructured test data from VAX computer system into RT-11 Operating System data format. This node is implemented by a system library routine and, therefore, it is not being elaborated any further (Figure B-14).

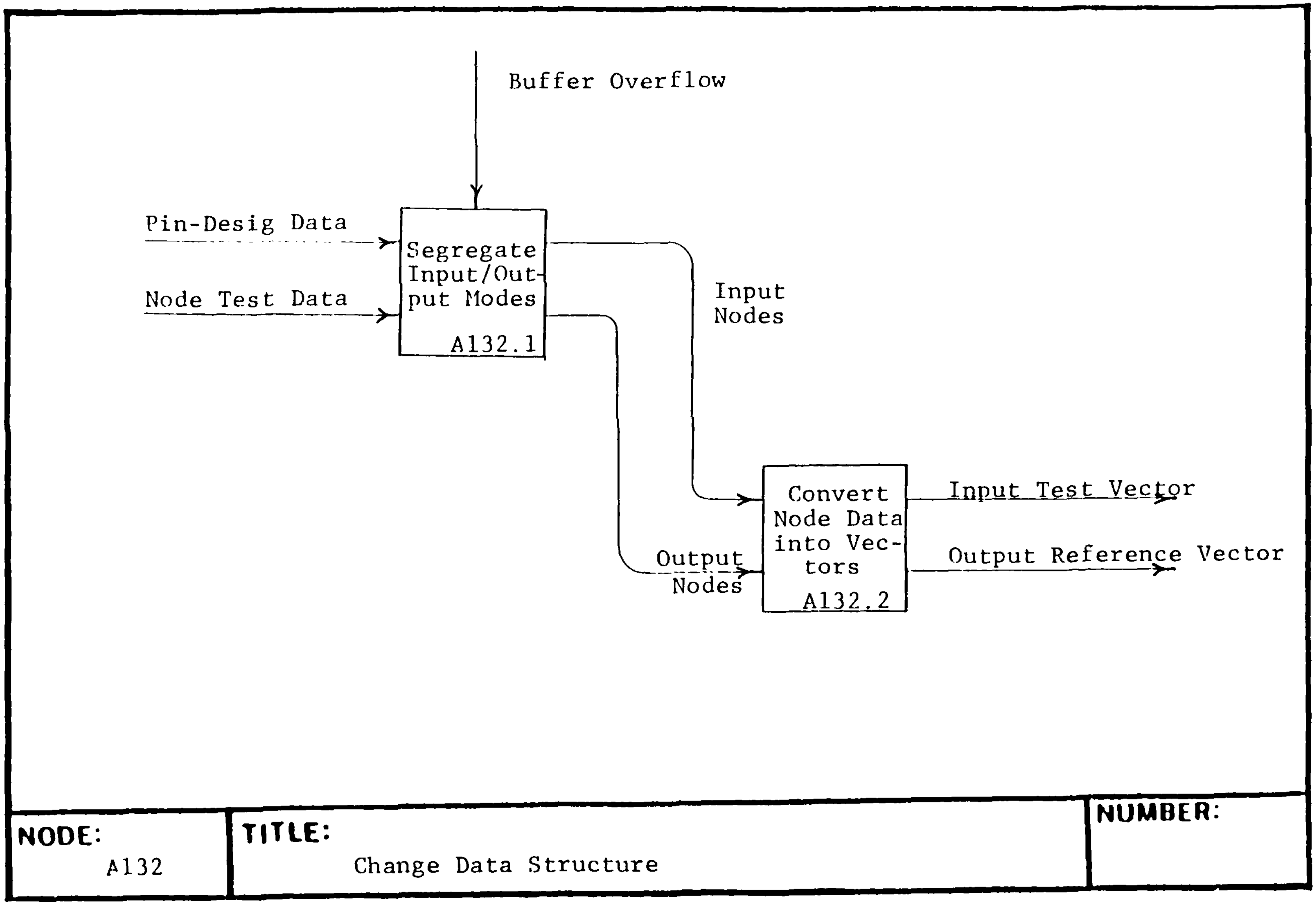


Figure B-13: Change Data Structure

Node A132: Change Data Structure

Abstract

This node segregates the nodes into two classes of input and output nodes. It converts the data associated with input nodes into input test vectors and data associated with output nodes into output reference vectors.

Change Data Structure

This node is decomposed into two functions as shown in Figure B-13. These functions are:

(i) Segregate Input & Output Nodes: This function classifies the test nodes into input and output nodes by consulting arrays of input pins and output pins respectively.

(ii) Convert Node Data into Vectors: This function converts node data contained in input nodes and output nodes into test vectors and reference vectors respectively. It considers all input nodes as an array, whose elements itself are character arrays. This function transposes this array to convert node data into test vectors.

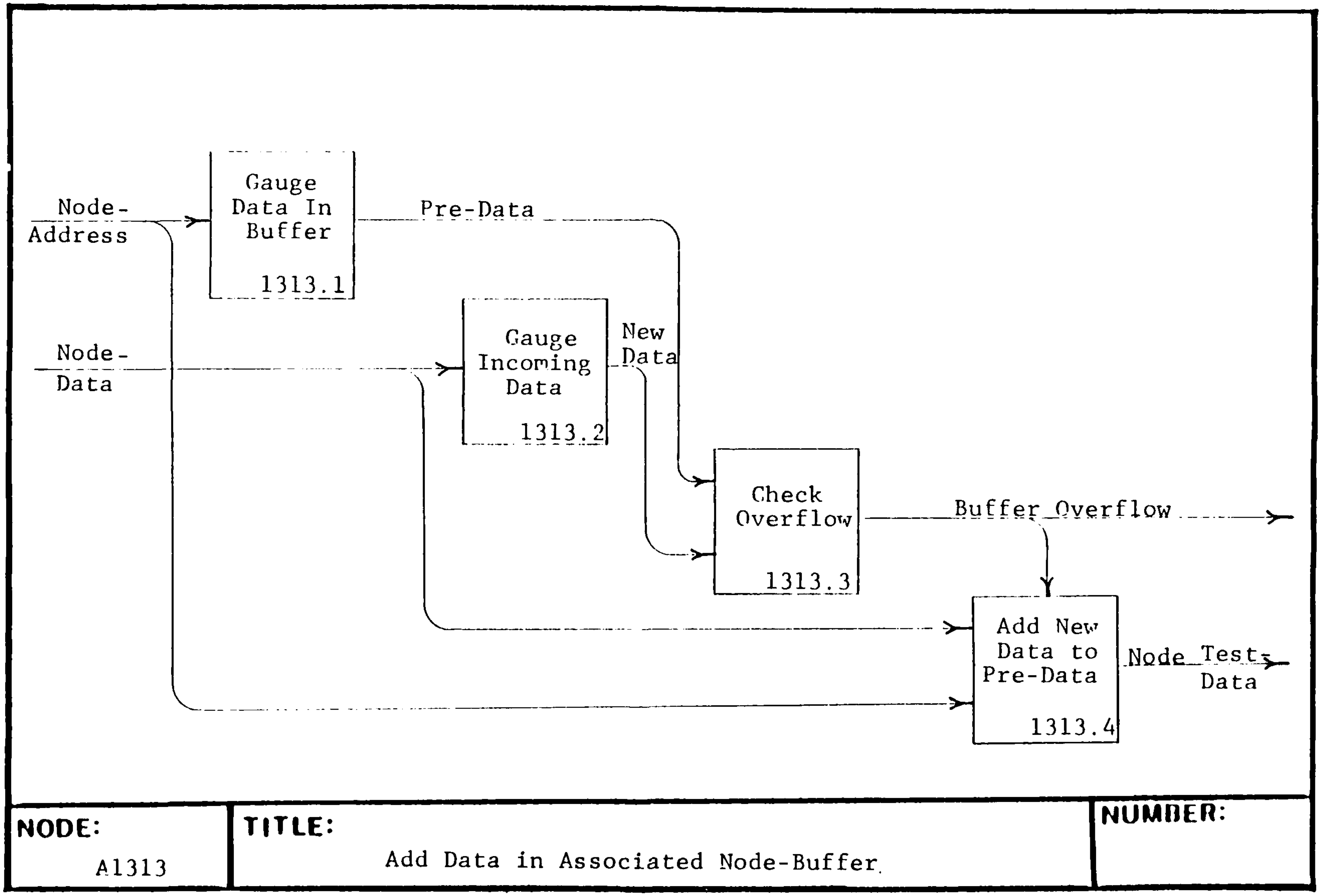


Figure B-12: Add Data in Associated Node-Buffer

NODE A1313: Add Data in Associated Node Buffer

Abstract

This node gauges the amount of data in a given "data line" and checks the empty space available in its associated buffer. It generates a buffer overflow signal if incoming data exceeds the available empty space or otherwise adds the new data to previous data.

Add Data in Associated Node Buffer

This node is decomposed into four functions as shown in Figure B-12. These functions are:

(i) Gauge Data in Buffer: This function checks the amount of data in a buffer -- it is implemented by a system library routine (strlen(Sl)) and will not be elaborated any further.

(ii) Gauge Incoming Data: This function checks the amount of data in the given "data line". It is implemented by a system library routine (strlen()) and will not be elaborated any further.

(iii) Check Overflow: This function, simply adds the amounts of new data and pre(vious)-data and compares it with buffer capacity. If the total data exceeds the buffer capacity, buffer overflow signal is generated.

(iv) Add New Data to Pre-Data: This function, if buffer overflow signal is not generated, adds new data to the buffer by a system library routine (strcat (S_1, S_2)).

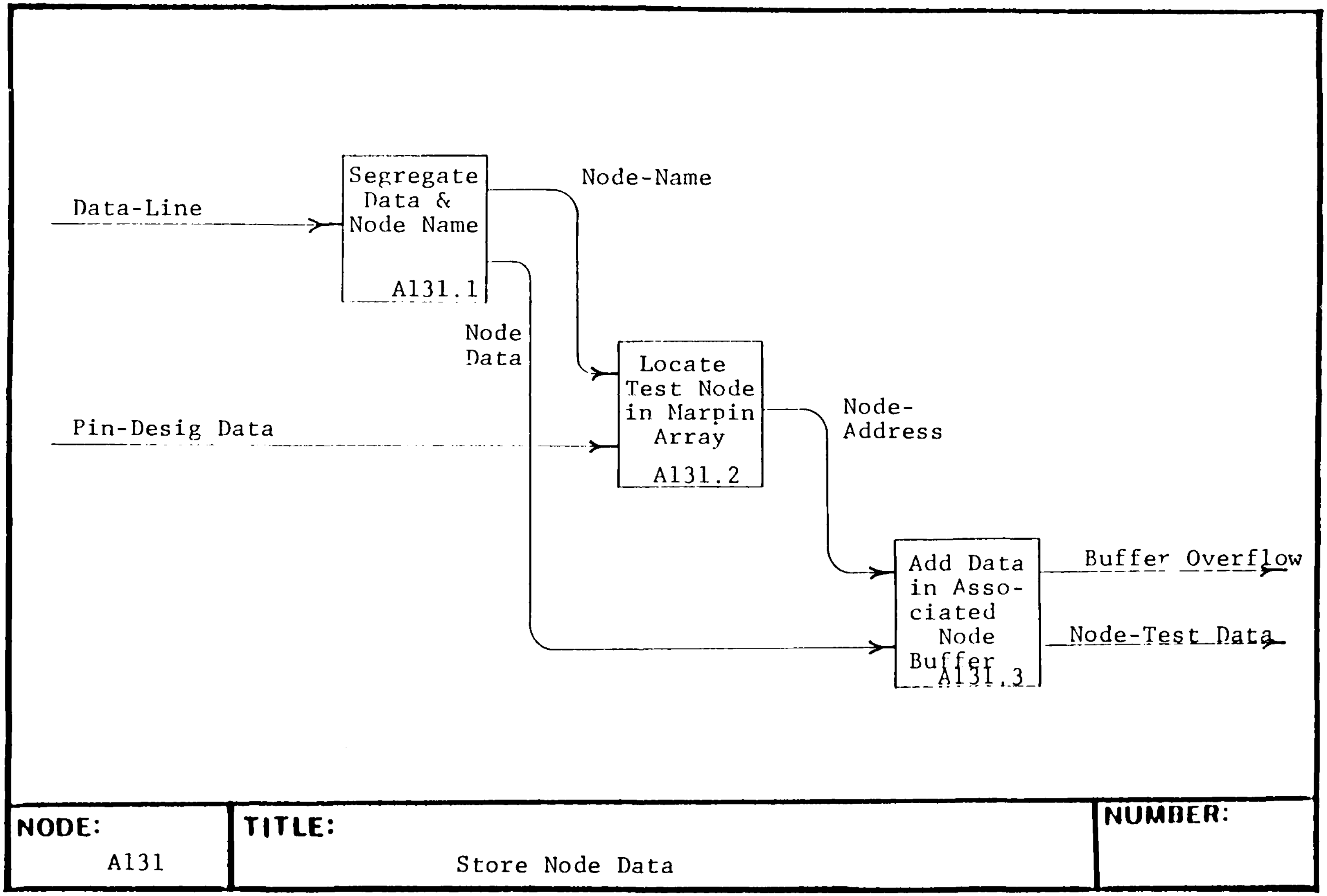


Figure B-11: Store Node Data

NODE A131: Store Node Data

Abstract

This node, on receiving a data line, adds the test data in the associated node buffer. In case of buffer overflow, it generates an error signal by which a function is activated to empty the buffer before adding any more data to the buffer.

Store Node Data

This node is decomposed into three major functions as showin in Figure B-11. These functions are:

(i) Segregate Data and Node Name: This function partitions a given data line into data-part and name-part by sensing the presence of ":" (colon-mark). An example of typical data-line is shown in Appendix-"F" (Page F-11).

(ii) Locate Test Node in Monpin-Array: This function scans the array of monitored pins and locates the pin, for which test data has been received in a given "data-line".

(iii) Add Data in Associated Buffer: This function gauges the incoming data and empty space in the associated buffer. Buffer overflow signal is generated if incoming data exceeds the capacity of buffer otherwise data is added to the specific buffer.

Node A22: Select Operating Option

Abstract

This node, receives option data from keyboard in response to different options made available to user in various menues, verifies their correctness and sets option flags to this effect.

Select Operating Option

This node is decomposed into three major functions as shown in Figure C-3. These functions are:

(i) Select Mode: This function, in response to system prompt receives IC nomenclature and preferred mode of operation data from keyboard. In addition, this function asks the user to input name of test file for "Auto" mode of operation.

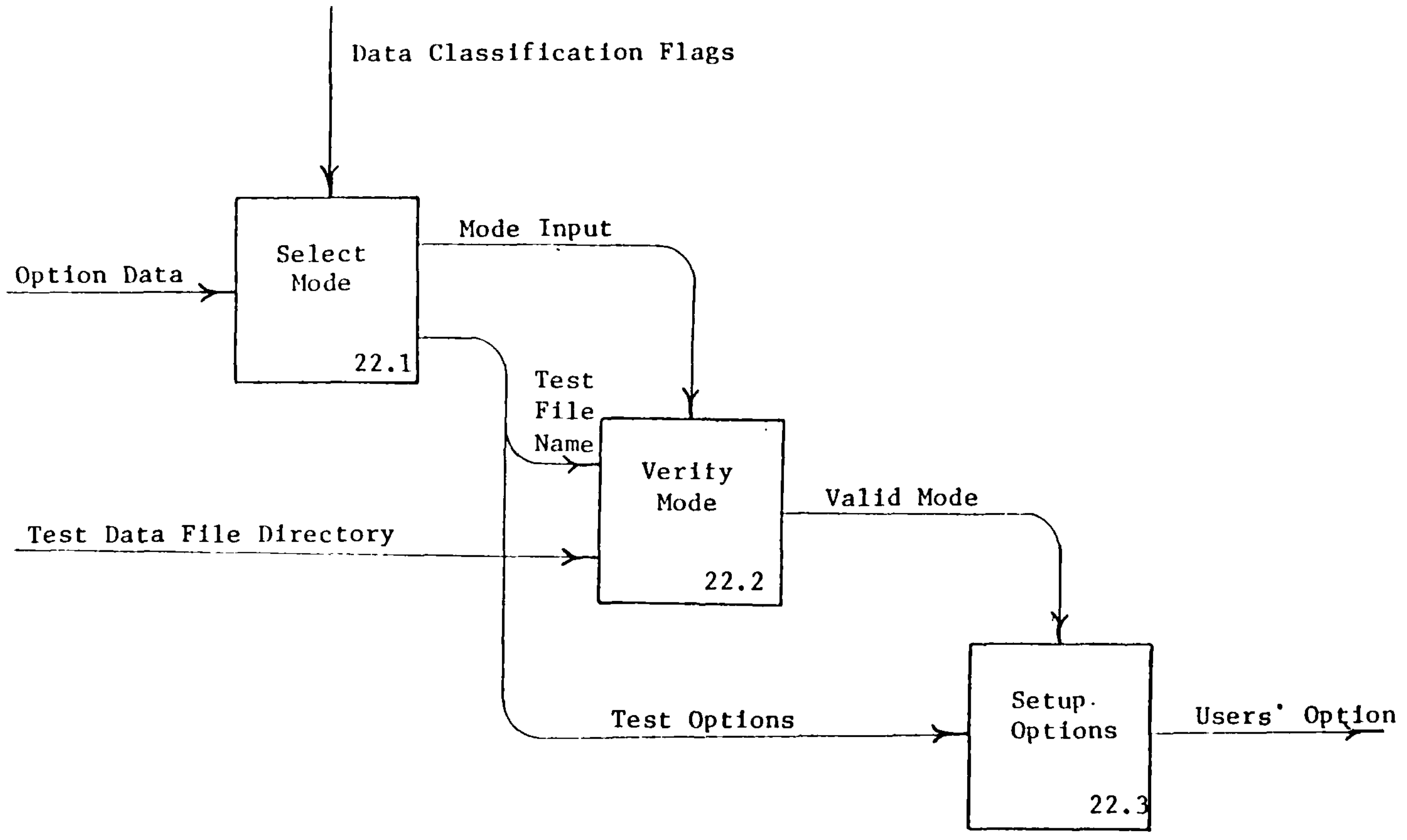
(ii) Verify Mode: This function checks the IC nomenclature and searches the test data files' directory with test file name. It generates an error for non-availability of test file, informs the users to this effect and asks for another selection.

(iii) Setup Options: This function sets various option flags to be true/false for particular selection after user has opted for one of the operating choices offered to him in system operating menus.

All user options and corresponding flags are listed below:

UOPM MANUAL mode. (abbreviated as MAN)
A UOPM10 MAN with single step execution and terminal output only.
B UOPM11 MAN with single step execution and terminal and file output.
C UPOM20 MAN with multi-step execution and Terminal output only.
D UPOM21 MAN with multi-step execution and terminal & file output

UOPA AUTO mode
A UOPA10 AUTO with "stop execution at first test failure" and terminal output only.
B UOPA11 AUTO with "stop execution at first test failure" and terminal & file output.
C UOPA21 AUTO with "stop execution after X-instruction" and terminal output only.
D UOPA22 AUTO with "stop execution after X-instructions" and terminal and file output.
E UOPA31 AUTO with "run whole test program printing all test failures" on terminal.
F UPOA32 AUTO with "run whole test program printing all test failures" on terminal and storage file.



NODE:	TITLE:	NUMBER:
A22	Select Operating Options	

Figure C-3: Select Operating Options

Node A221: Select Mode

Abstract

This node in response to system prompt receives from keyboard the preferred mode of operation. For "Auto" mode of operation, it asks the user to input name of respective test-data file.

Select Mode

This node is decomposed into four functions as shown in Figure C-4. These functions are:

(i) Get Mode: This function receives a character "A" or "M" from keyboard for Auto or Manual mode of selection as preferred mode of operation.

(ii) Set Flag for Auto: This function sets respective flags to be true for "Auto" mode of operation or "Manual" mode of operation.

(iii) Prompt for Name of Test File: This function is active only if "Auto" mode is selected. It prompts the user to input name of file which contains the test data to simulate ICUT and reference data to verify correctness of results.

(iv) Store Test-File Name: This function stores name of test-file in a specific buffer for future reference. In "C" language, this is readily implemented by system library function (scanf (S)).

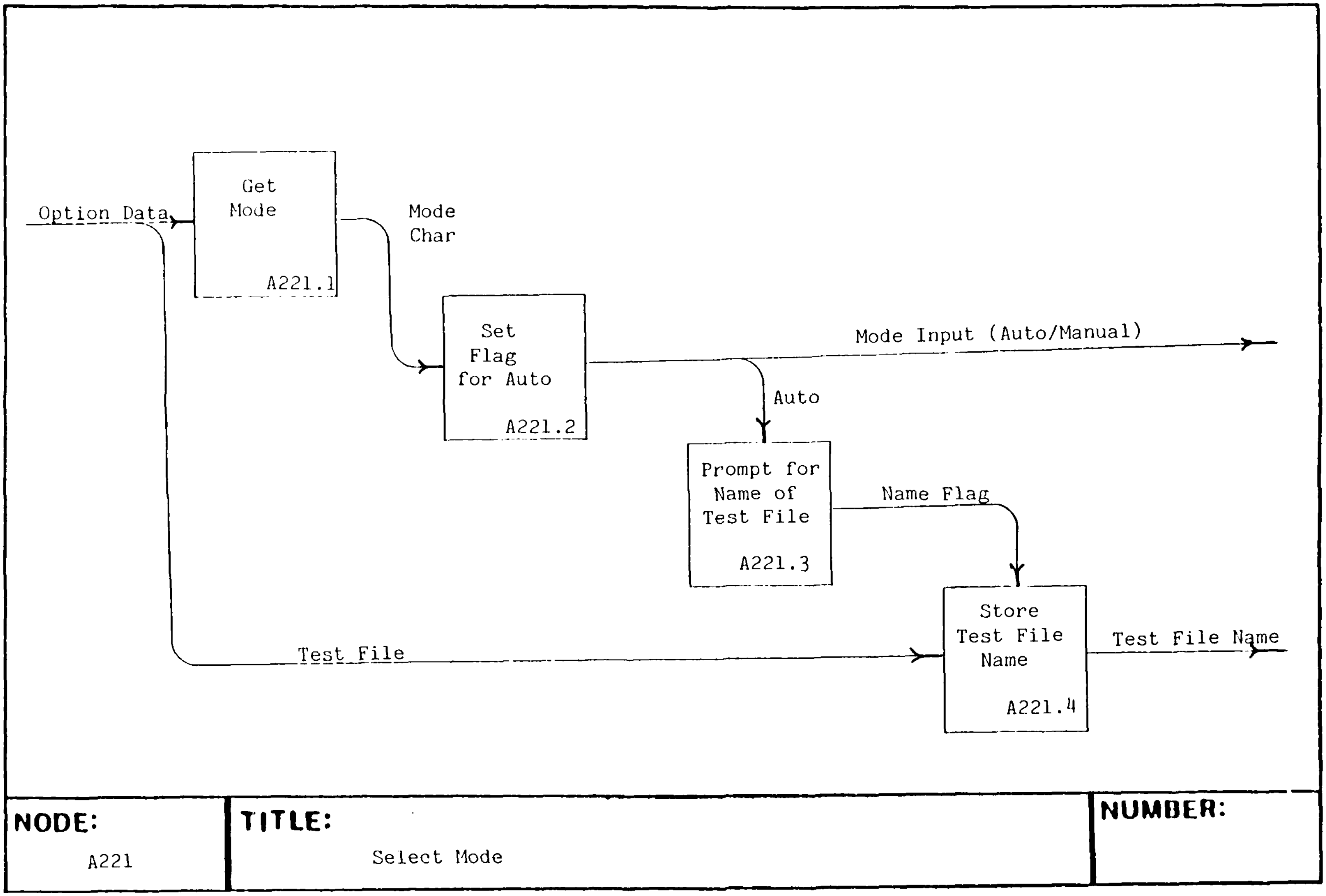


Figure C-4: Select Mode

Node A222: Verify Mode

Abstract

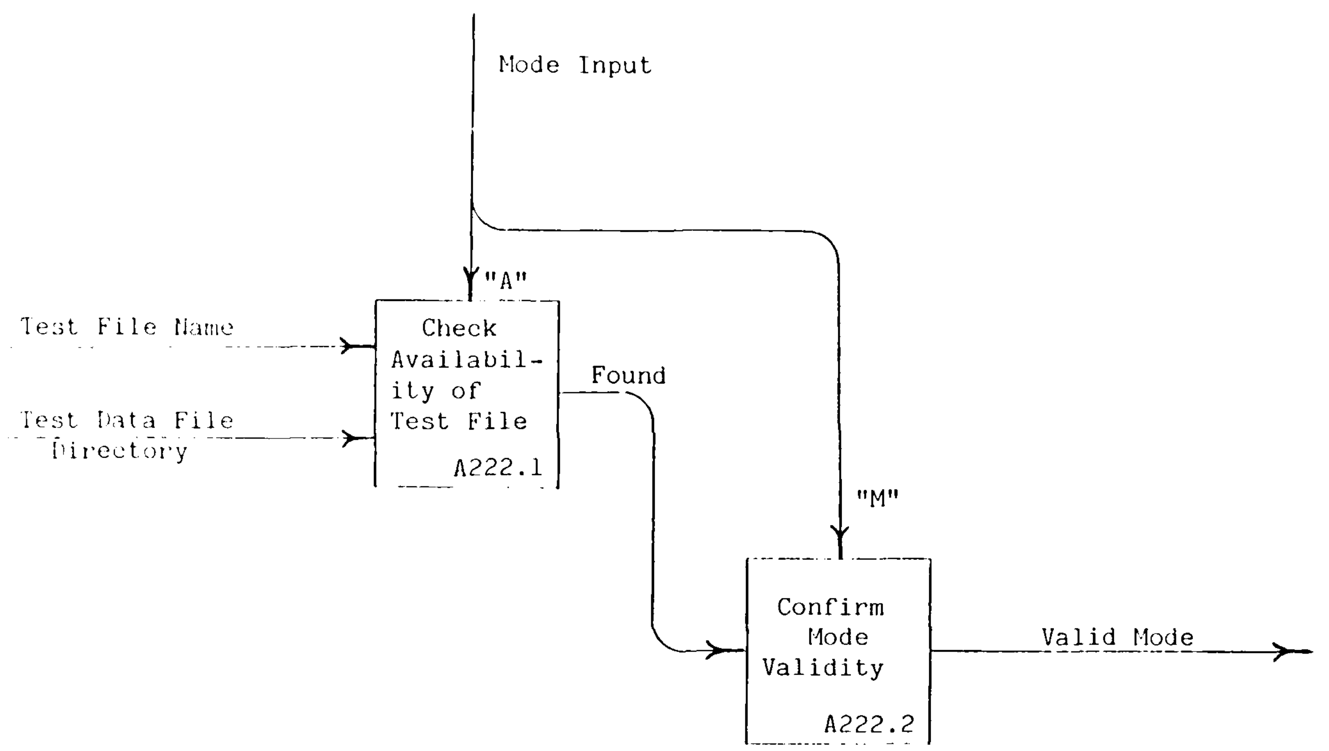
This node, sets flag for "valid mode" if test data file is available for Auto mode of selection. The mode is always valid for "Manual" mode of operation.

Verify Mode

This node is decomposed into two functions as shown in Figure C-5. These functions are:

(i) Check Availability of Test File: This function scans the file directory of a disk to check availability of test data file, whose name has been input. This function is readily implemented in "C" language by system function (oper (...)), which returns a zero value for unsuccessful access to a given file.

(ii) Confirm Mode Validity: This function sets valid-mode flag to be true in "Manual" mode and also in "Auto" mode if system function (oper(...)) returns a non-zero value after successfully accessing the given file.



NODE: A222

TITLE: Verify Mode

NUMBER:

Figure C-5: Verify Mode

Node A223: Setup Options

Abstract

This node prompts user to select one of the offered sub-node of operations from a given menu. This node checks the syntax of the particular selection and sets option flag to that effect.

Setup Option

Thisn ode is decomposed into four functions as shown in Figure C-6. These functions are:

(i) Prompt User to Select Option: This function, offers one of the broad categories "Auto or Manual" option menu to user for making a selection. The details of all available option are described in description of Node A223.

(ii) Get Option Response: This function reads thekeyboard input, offered in response to the select-option menu.

(iii) Check Option Syntax: This function, checks the syntax of input, for it to be within expected range of response. This function repeats itself until valid response is received from user.

(iv) Set Option Flag: This function, on valid selection of particular sub-mode of operation, sets the respective flag to be true.

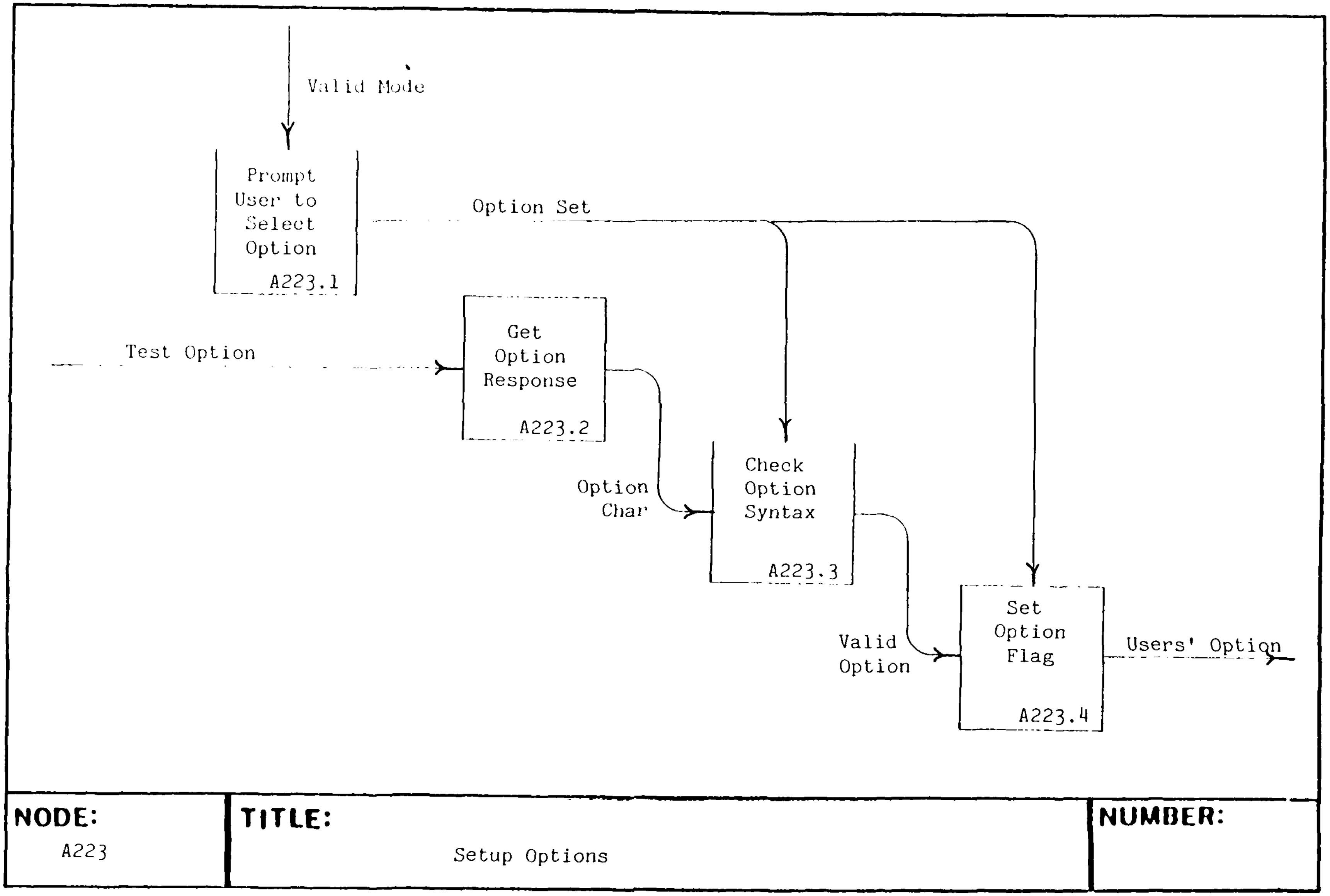


Figure C-6: Setup Options

NODE A23: Setup Reference Table

Abstract

This NODE selects one of the prestored tables depending on physical characteristics of an ICUT. These reference tables contain information to set one to one correspondence between IC Pins and IC Tester pins for an IC of particular physical characteristics. It then prompts the user to provide information about input/output and other significant pins of ICUT. This node also stores this data to correlate ICpin numbers in a test vector to their physical location on IC Tester for effecting a test simulation.

Setup Reference Tables

This node is decomposed into six functions as shown in Figure C-7. These functions are:

(i) Select Appropriate Table: This function on receiving an input from keyboard, in response to a menu which prompts the user to select one of the five IC characteristics befitting the ICUT, makes the address of corresponding table available to the program. Five different tables for ICs with following characteristics have been prestored in the memory: 14, 16, 20, 24, 40pin dual-in-line packages. These tables correlate the IC pin numbers to IC tester pins when ICUT is positioned on the IC tester in a manner that pinl of ICUT coincides with the top-left pin of IC tester.

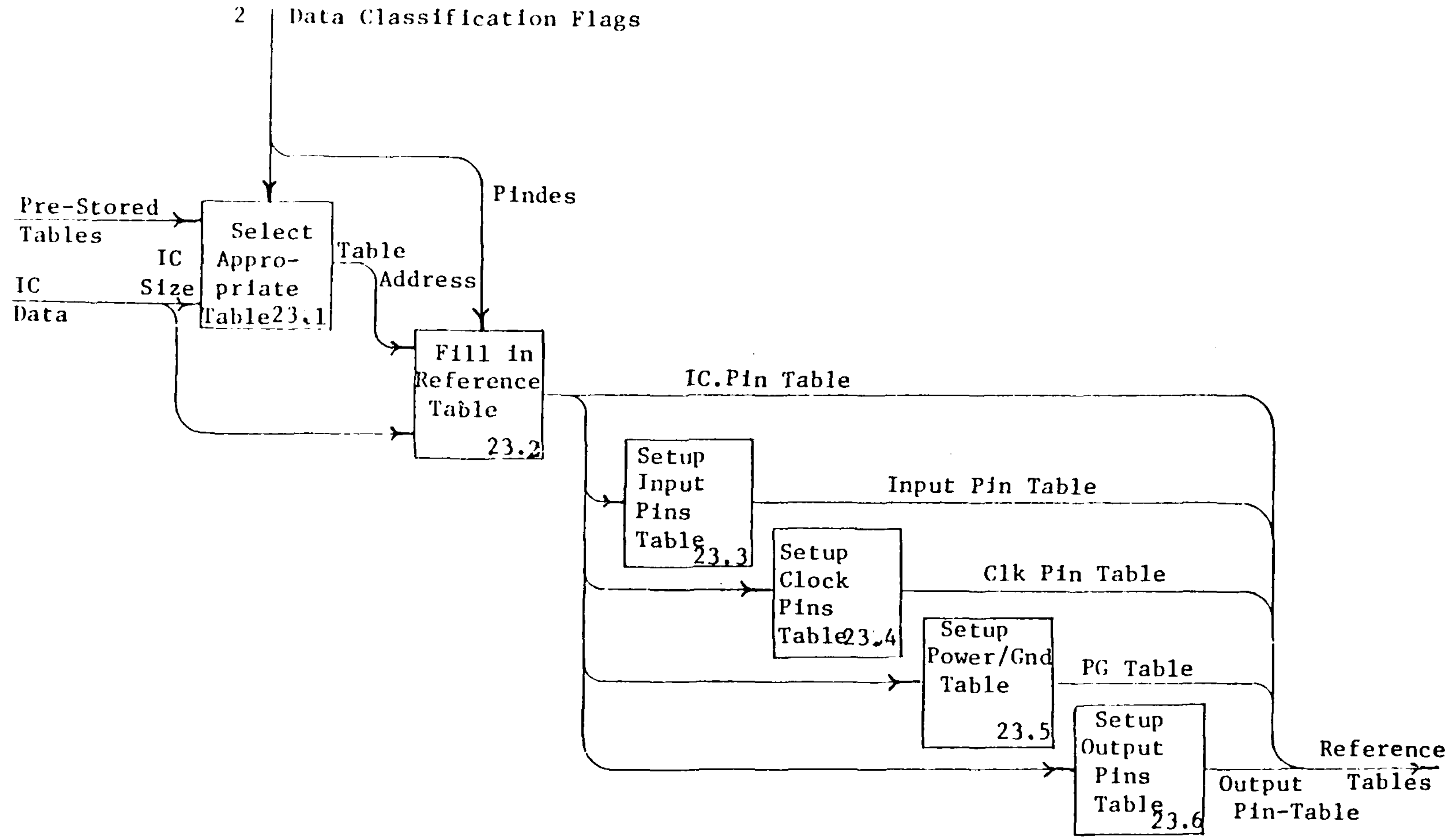
(ii) Fill in Reference Table: This function prompts the user to fill in complete reference table by inputting pin designation and class of each pin. (e.g., to provide information like pinl = k-input "I", pin4 = phi2 "C", pinl4 = Vcc "P" and pin7 = Gnd "G", etc.)

(iii) Setup Input Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "input" pins.

(iv) Setup Clock Pins Table: This function, scans the reference taole after it has been filled up by the user, and sets up another array for pins marked as "output" pins.

(v) Setup Power/Ground Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "power/ground" pins.

(vi) Setup Output Pins Table: This function, scans the reference table after it has been filled up by the user, and sets up another array for pins marked as "output" pins.


2
Data Classification Flags
Pre-Stored Tables
IC Size
IC Data
Select Appro-priate Table 23.1
Table Address
Pindes
Fill in Reference Table 23.2
IC.Pin Table
Setup Input Pins Table 23.3
Input Pin Table
Setup Clock Pins Table 23.4
Clk Pin Table
Setup Power/Gnd Table 23.5
PG Table
Setup Output Pins Table 23.6
Output Pin-Table
Reference Tables

NODE:	TITLE:	NUMBER:
A23	Setup Reference Tables	

Figure C-7: Setup Reference Tables

Node A231: Select Appropriate Table

Abstract

This node, in response to a given menu receives the input, validate its syntax, and points to the address of appropriate pre-stored table.

Select Appropriate Table

This node is decomposed into four functions as shown in Figure C-8. These functions are:

(i) Prompt User to Mark IC-Characteristics: This function offers a menu containing varied combination of IC Characteristics (size, shape of IC and number of pins, i.e., 20 pin dual-in line, 40 pin square, etc.) and user is asked to make selection which matches the IC to be tested.

(ii) Get IC-Characteristics: This function gets the character selection made by user in response to system menu.

(iii) Check Syntax of Selection: This function checks the input for it to be within expected range of response.

(iv) Locate Appropriate Table: This function, correlates the valid selection with the address of appropriate table. This correspondence has been set aprior by the programmer.

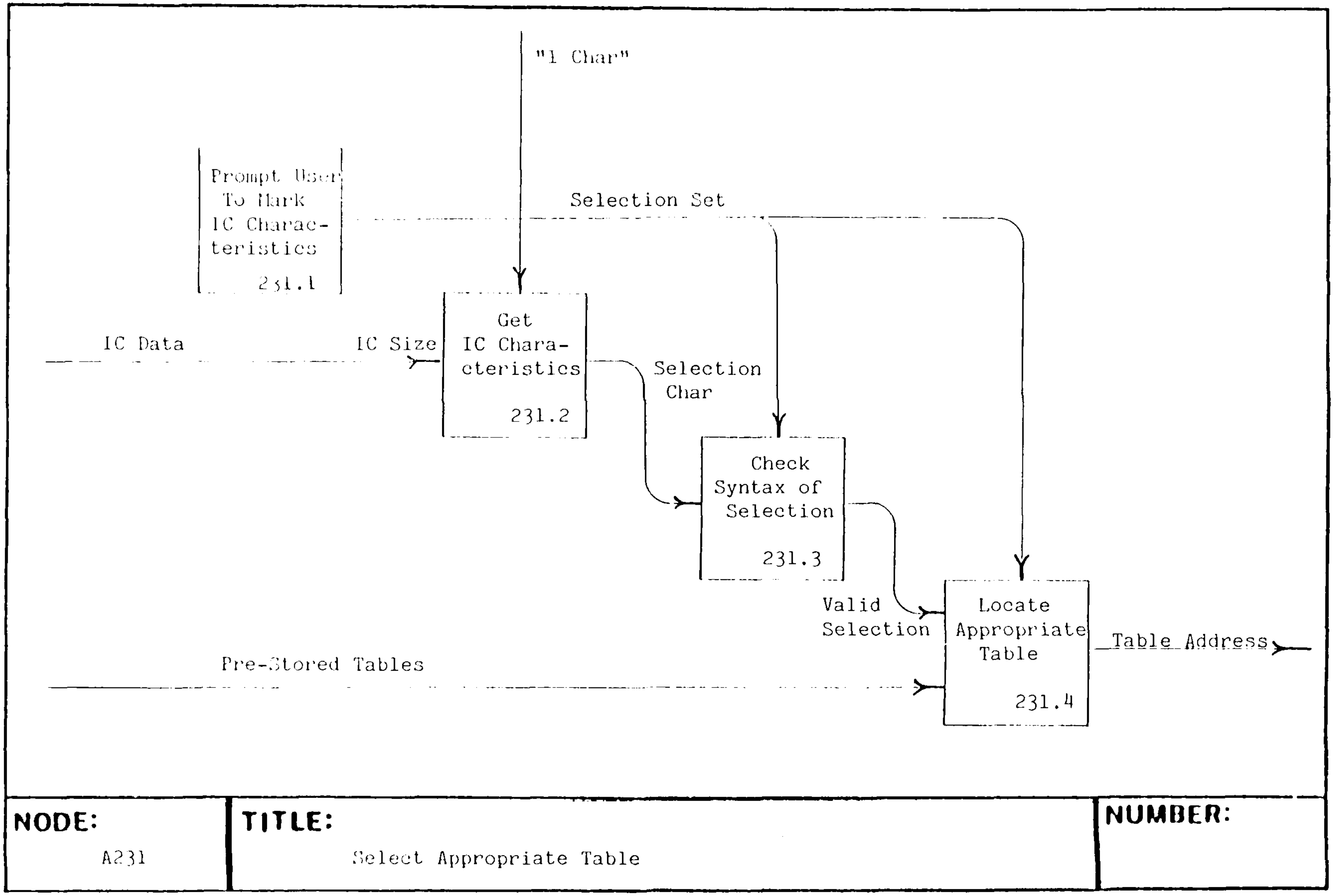


Figure C-8: Select Appropriate Table

Node A232: Fill-In Reference Table

Abstract

This node prompts user to input pin designations (i.e., O1, Vcc, reset, etc.) and pin-class (clock, power, input, output, etc.) for all pins and stores the data received from keyboard.

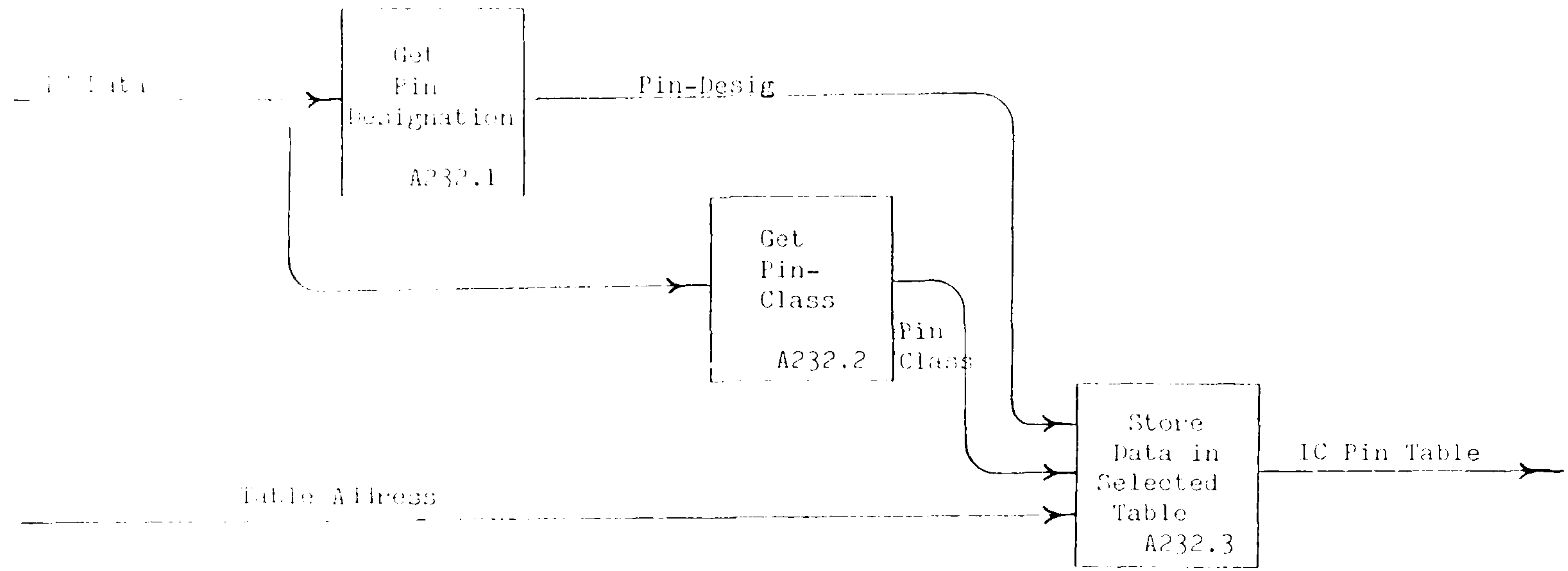
Fill-In Reference Table

This node is decomposed into three functions as shown in Figure C-9. These functions are:

(i) Get designations: This function asks the user to input designated name of a pin of the IC by giving a number (for example, input name of pin 1 = ?).

(ii) Get Pin-Class: This function asks the user to input designated class of the pin (P = power, G = Gnd, I = input, O = output, K = clk, X = don't care).

(iii) Store Data in Selected Table: This function stores the pin designation and pin-class in the selected table. This function is readily implemented in "C" language by system library function (strcpy (S_1, S_2)).



NODE:	TITLE:	NUMBER:
A232	Fill in Reference Table	

Figure C-9: Fill-in Reference Table

```
main()

 int i, err=1, number=0;
char  inname[], outname[], textline[MAXL];
char  *makenull, *line[MAXL], *textline_ptr;
TABLE  table, *table_ptr;
PARAMETER  flag, *flag_ptr;
VECDATA sim_array[MAXL], *simptr;
IOPIN monpin_table[40];
DATA  ldata[20], clk_seq[10],outpin_table[20], hlarray[10];
table_ptr = &table;
flag_ptr  = &flag;
simptr    = sim_array;
table.monpin = monpin_table;
table.outpin = outpin_table;
table.inpin  = ldata;
table.clkpin = clk_seq;
table.hinode = hlarray;
textline_ptr = textline;
makenull = "";

 initialize(table_ptr,flag_ptr);
printf(" ENTER NAME OF ESIM ..testdata.. FILE :\n");
scanf("%24s",inname);
infile = fopen (inname, "r");

if ( infile == NULL
   printf( "Couldn't open %s for reading.\n",inname);
else

   printf(" ENTER NAME OF STORAGE FILE :\n");
   scanf("%24s",outname);
   outfile = fopen ( outname, "a" );
    if ( outfile == NULL )
     printf( "Couldn't open %s for writing.\n",outrame);
   else

    while ( feof(infile)== 0 )
    {
     fgets(line,MAXL,infile);
     if ( line != NULL )
     {
       copy_string(line, textline_ptr);
        if ( not_blank_line(textline) )

         if ( alphabetic(textline[0]) )

           tabulate_cmd_data(table_ptr,flag_ptr,textline,simptr,outfile);
            if ( flag.pin_chg )
            {
             restructure_testdata(table_ptr,flag_ptr,simptr);
             append_file(flag_ptr,table_ptr,simptr,outfile);
             flag.pin_chg = 0;
             tabulate_cmd_data(table_ptr,flag_ptr,textline,simptr,outfile);

          }
        if ( textline[0] == '[illegible]' )
           restructure_testdata(table_ptr,flag_ptr,textline,simptr,outfile);

     }
    }
    flag.final = 1;
    restructure_testdata(table_ptr,flag_ptr,textline,simptr,outfile);
```

```
/*************************************************************************/
 *                                                                      */
 *                   E X T R A C T    T E S T    D A T A               */
 *                                                                      */
 ************************************************************************/
 *                                                                      */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                     */
 * THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE  */
 *                     FOR INPUT TO STANFORD IC TESTER                  */
/* THESIS ADVISOR:  COL. H.CARTER                                       */
 * DESIGN DATE:  JULY 2 ,1984          LAST UPDATE: November 4, 1984    */
 * DACT REFERENCE: A1                                                   */
 *----------------------------------------------------------------------*/
 * DESCRIPTION: This module controls the activities of all other modules.  */
 * It checks for valid input/output files; gets command and test data from */
 * input file; categorises all test nodes into three categories of input/  */
 * output/clock pins by interpreting the command lines. It also restructures */
 * the incoming node test data into test vectors and writes it out in a    */
 * given file.                                                          */
 *----------------------------------------------------------------------*/
 * PSEUDO CODE:                                                         */
 *               if ( input & output files are valid )                  */
 *                 {                                                    */
 *                   while ( not eof(input_ESIM file))                  */
 *                   {                                                  */
 *                      get_next_line(ESIM file);                       */
 *                      classify_rcvd_line(line);                       */
 *                      if (command line(line)                          */
 *                         tabulate_command_data(line);                 */
 *                      else                                            */
 *                         tabulate_test_data(line);                    */
 *                   }                                                  */
 *                   restructure_node_data();                           */
 *                 }                                                    */
 *----------------------------------------------------------------------*/
 * CALLING ARGUMENTS:                                                   */
 *                          None                                        */
 *                                                                      */
 *----------------------------------------------------------------------*/
 * FILES USED:                                                          */
 *  infile   input ESIM file, which contains node designations and test data.*/
 *  outfile: output file, in which restructured test vectors from the given  */
 *             node data are stored for later consumtion.               */
 *----------------------------------------------------------------------*/
 * MODULES CALLED:                                                      */
 *       initialize();                                                  */
 *       tabulate_cmd_data();                    tabulate_test_data();  */
 *       restructure_test_data();                                       */
 ************************************************************************/
# include "def.h"
# include <stdio.h>

FILE    *infile, *outfile;
```

```
*****************************************************************************/
*                                                                         */
*                        D E F I N I T I O N                              */
*                                                                         */
*****************************************************************************/
*                                                                         */
* STUDENT NAME  :  SQN.LDR.SALEEM                                         */
* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE     */
*                        FOR INPUT TO STANFORD IC TESTER                  */
* THESIS ADVISOR:  COL. H.CARTER                                          */
* DESIGN DATE:  JULY 2 ,1984                  LAST UPDATE: November 3, 1984 */
* SADT REFERENCE:   common to all other modules.                          */
*-------------------------------------------------------------------------*/
* DESCRIPTION: This module defines various constants and structures for   */
*  their use in the subsequent programs.                                  */
*                                                                         */
*-------------------------------------------------------------------------*/
* PSEUDO CODE:                                                            */
*                                   -                                     */
*-------------------------------------------------------------------------*/
* CALLING ARGUMENTS:                                                      */
*                                   -                                     */
*-------------------------------------------------------------------------*/
* MODULES CALLED: This module is a part of all other modules.             */
*                                                                         */
*****************************************************************************/
# define MAXL 1ØØØ
# define NOREF 99
# define DEBUG Ø

typedef  char   LABEL[2Ø];
typedef  char   TESTDATA[MAXL];

typedef struct
        {
        LABEL     name;
        char      class;
        int       class_ref;
        TESTDATA  datastg;
        }ICPIN;

typedef struct
        {
        LABEL     pin_desig;
        int       monref;
        TESTDATA  pin_data;
        }DATA;

typedef struct
        {
        LABEL   test_input;
        LABEL   ref_output;
        int     follow;
        int     change;
        }VECDATA;


 typedef struct
        {
        ICPIN   *monpin;
        DATA    *outpin;
        DATA    *inpin;
        DATA    *clkpin;
        DATA    *hlnode;
        }TABLE;


 typedef struct
        {
        int     *NUM_MONPIN;
        int     *inal;
        int     in_count;
        int     clk_count;
        int     out_count;
        int     second_time;
        int     init_again;
        int     pin_chg;
        int     cstg_count;
        int     overflow;
        int     hlcmd;
        int     hlcount;
        int     start;
        int     prn_count;
        }PARAMETER;
```

Appendix "D"

Program Listing of "Extract Test Data" Function

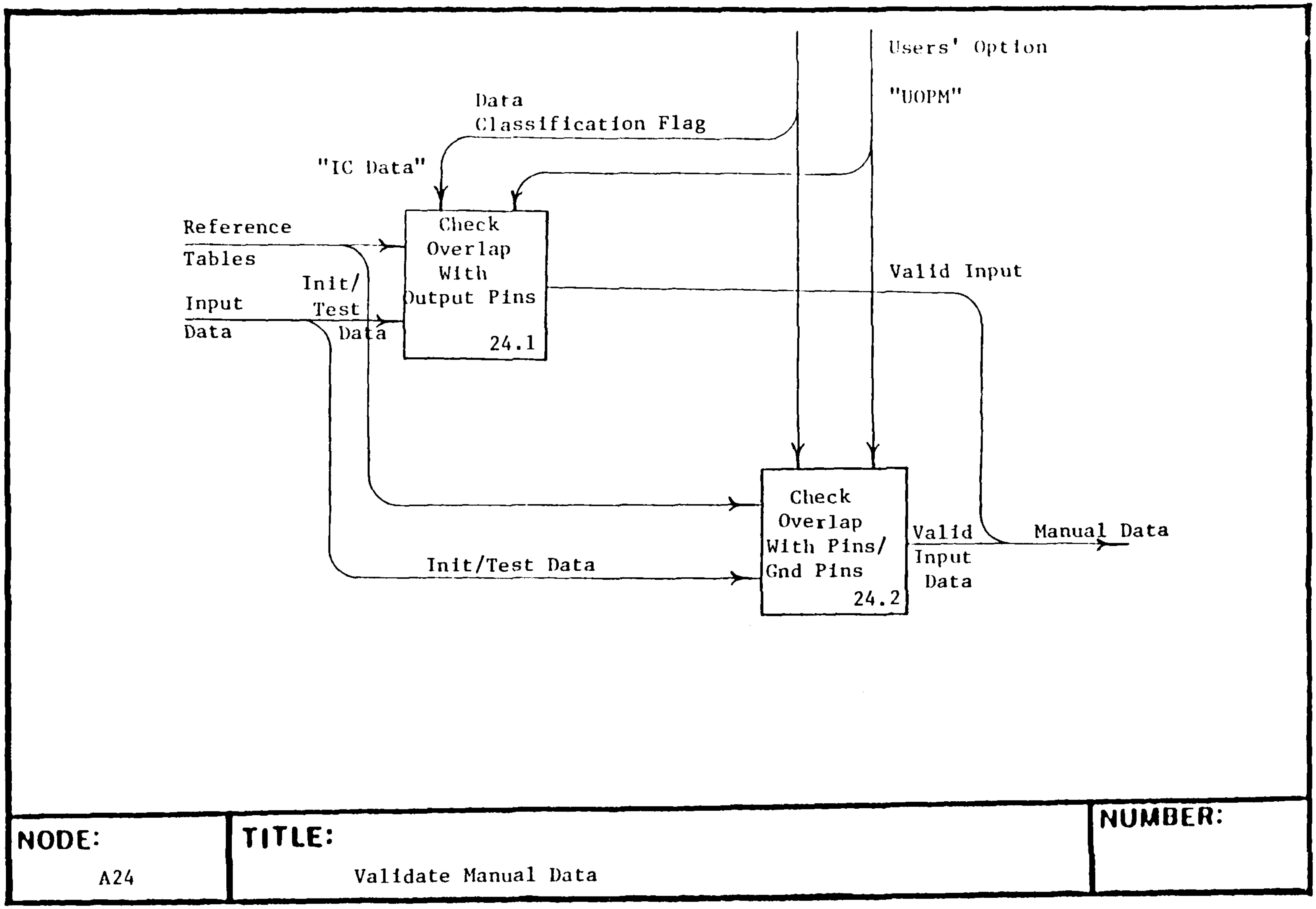


Figure C-14: Validate Manual Data

Node A24: Validate Manual Data

Abstract

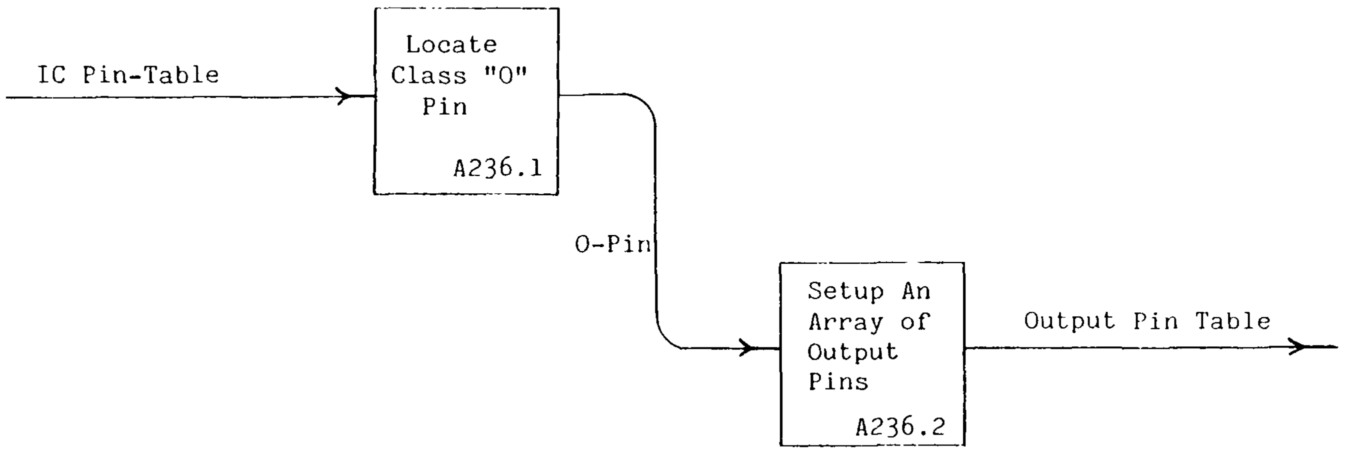
This node is activated only in manual mode of operation. It checks for any overlap of input test data over designated output/power/ground pins.

Validate Manual Data

This node is decomposed into two functions as shown in FigureC-14. These functions are:

(i) Check Overlap with Output Pins: This function carries out bitwise comparison of test vector to check if any pin designated as output pin is not simulated. It generates an error on detecting such an overlap and prompts user tomodify his input data. This function is activated only in "Manual" mode of operation.

(ii) Check Overlap with Power/Ground Pins: This function carries out bitwise comparison of test vector to chewck if any pin designated as power or ground pin isn ot simulated. It generates an error on detecting such an overlap and prompts user to modify his input data. This function is activated only in "Manual" mode of operation.



NODE: A236	**TITLE:** Setup Output Pins' Table	**NUMBER:**

Figure C-13: Setup Output Pins' Table

Node A236: Setup Output Pins Table

Abstract

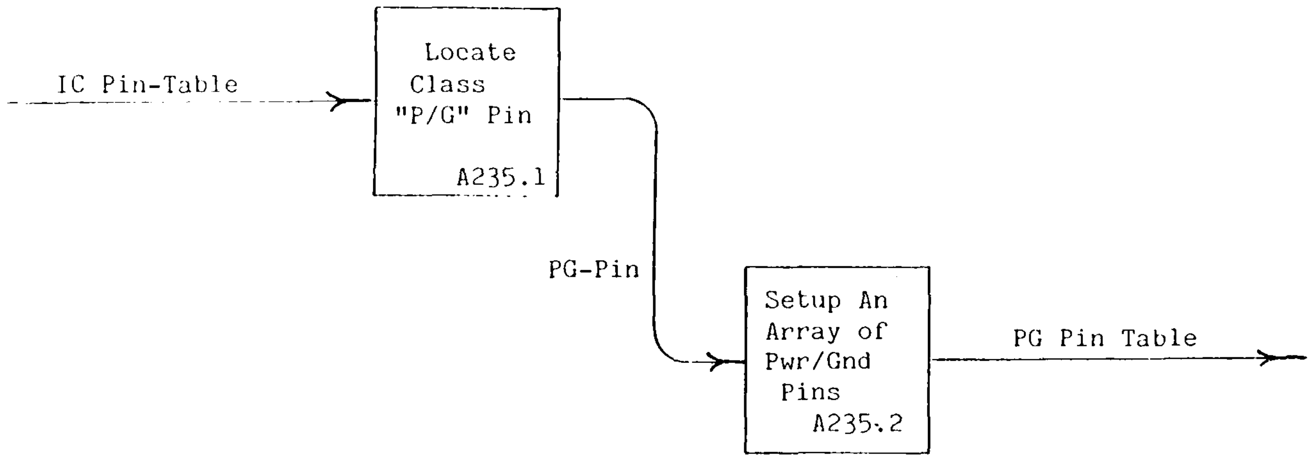
This node scans the IC pin table after it has been filled in with pin-desig and pin class for all pins. It searches the whole table to segregate "output - 0" class of pins and sets up a separate array for quick reference.

Setup Output Pin Table

This node is decomposed into two functions as shown in Figure C-13. The functions are:

(i) Locate Class "O" Pin: This function scans the class-field of IC pin table and point out if it matches with "O".

(ii) Setup an Array of Output Pins: This function stores the first matched pin (from para i) as zeroth element of an array and adds the subsequent matching pins. The arrays are defined aprior in the system by programmer.



NODE: A235

TITLE: Setup Pwr/Gnd Pins' Table

NUMBER:

Figure C-12: Setup Power/Ground Pins' Table

Node A235: Setup Pwr/Gnd Pins Table

Abstract

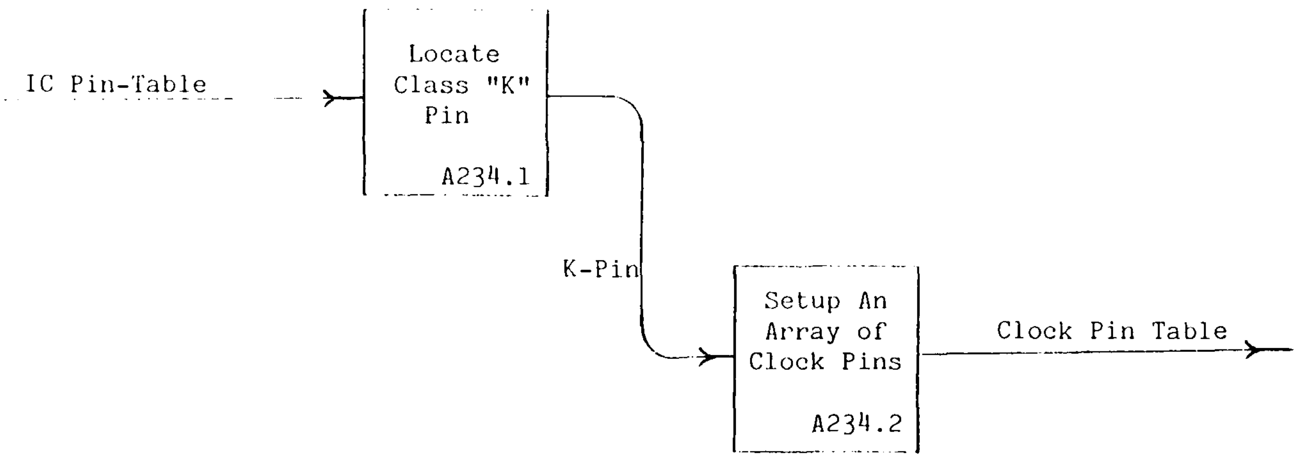
This node scans the IC pin table, after it has been filled in with pin-design and pin class for all pins. It searches the whole table to segregate "Power/Ground-P/G" class of pins and sets up a separate array for quick reference.

Setup Power/Ground Pin Table

This node is decomposed into two functions as shown in Figure C-12. The functions are:

(i) Locate Class "P/G" Pin: This function scans the class-field of IC pin table and point out if it matches with "P/G".

(ii) Setup an Array of Pwr/Gnd Pins: This function stores the first matched pin (from para i) as zeroth element of an array and adds the subsequent matching pins. The arrays are defined aprior in the system by programmer.



NODE: A234	**TITLE:** Setup Clock Pins' Table	**NUMBER:**

Figure C-11: Setup Clock Pins' Table

Node A234: Setup Clock Pins Table

Abstract

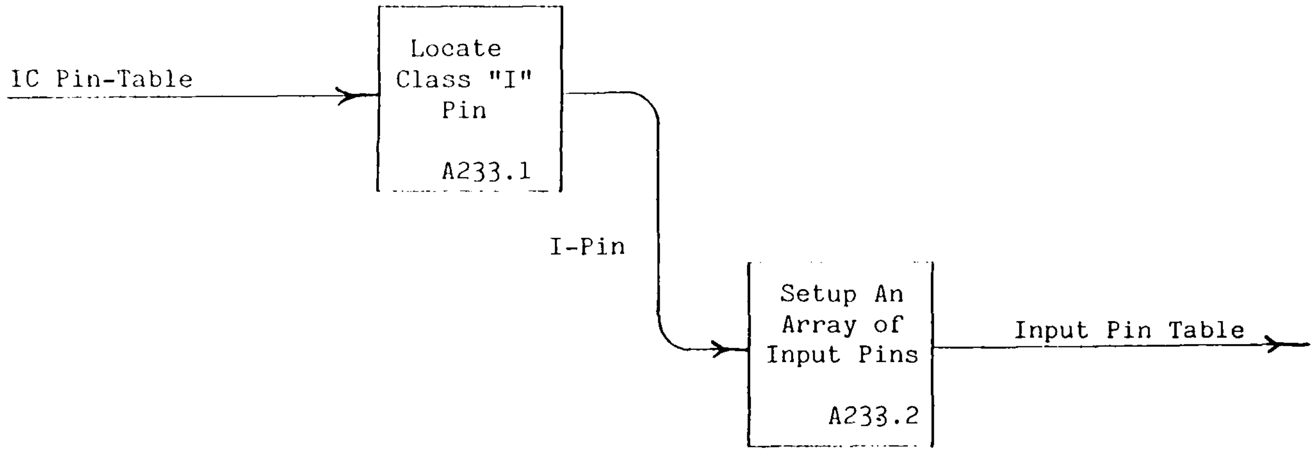
This node scans the IC pin table, after it has been filled in with pin-desig and pin class for all pins. It searches the whole table to segregate "clock - k" class of pins and sets up a separate array for quick reference.

Setup Clock Pin Table

This node is decomposed into two functions as shown in Figure C-11. The functions are:

(i) Locate Class "K" Pin: This function scans the class-field of IC pin table and point out if it matches with "K".

(ii) Setup an Array of Clock Pins: This function stores the first matched pin (from para i) as zeroth element of an array and adds the subsequent matching pins. The arrays are defined aprior in the system by programmer.



NODE: A233

TITLE: Setup Input Pins Table

NUMBER:

Figure C-10: Setup Input Pins' Table

Node A233: Setup Input Pins Table

Abstract

This node scans the IC pin table, after it has been filled in with pin-desig and pin class for all pins. It searches the whole table to segregate "input - I" class of pins and sets up a separate array for quick reference.

Setup Input Pin Table

This node is decomposed into two functions as shown in Figure C-10. These functions are:

(i) Locate Class "I" Pin: This function scans the class-field of IC pin table and point out if it matches with "I".

(ii) Setup an Array of Input Pins: This function stores the first matched pin (from paragraph i) as zeroth element of an array and adds the subsequent matching pins. The arrays are defined aprior in the system by programmer.

```
/************************************************************************/
/*                                                                      */
 *                         I N I T I A L I Z E                         */
 *                                                                      */
*************************************************************************/
 *                                                                      */
 * STUDENT NAME   :  SQN.LDR.SALEEM                                     */
 * THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE  */
 *                     FOR INPUT TO STANFORD IC TESTER                  */
 * THESIS ADVISOR:  COL. H.CARTER                                       */
 * DESIGN DATE:  JUL 2 ,1984            LAST UPDATE: November 4, 1984   */
 * CALL REFERENCE:  --                                                  */
 *----------------------------------------------------------------------*/
 * DESCRIPTION: This module initializes all variables defined in the main */
 *          program.                                                    */
 *                                                                      */
 *----------------------------------------------------------------------*/
 *                                                                      */
 *                           -                                          */
 *----------------------------------------------------------------------*/
 * CALLING MODULE:                                                      */
 *                           -                                          */
 *----------------------------------------------------------------------*/
 * MODIFIED VALUES:                                                     */
 *                           -                                          */
*************************************************************************/
initialize(tptr,fptr)
 TABLE       *tptr;
 PARAMETER   *fptr;
{

int i:
ICPIN *entry:
DATA   *inptr, *clkptr, *outptr;

   entry = (tptr)->monpin:
   outptr= (tptr)->outpin:
   inptr = (tptr)->inpin:
   clkptr= (tptr)->clkpin:
   (fptr)->NUM_MONPIN=Ø;
   (fptr)->final=Ø;
   (fptr)->in_count=Ø:
   (fptr)->clk_count=Ø;
   (fptr)->out_count=Ø;
   (fptr)->second_time=Ø;
   (fptr)->init_again=Ø;
   (fptr)->pin_chg=Ø:
   (fptr)->dstg_count=Ø;
   (fptr)->overflow = Ø;
   (fptr)->nlcount = Ø;
   (fptr)->start = 1:
   (fptr)->nlcmd = Ø:
   (fptr)->prn_count = Ø;
}
```

```
/****************************************************************************/
/*                                                                          */
/*                      F U N C T I O N - F I L E                           */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                         */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE      */
/*                     FOR INPUT TO STANFORD IC TESTER                      */
/* THESIS ADVISOR:  COL. H.CARTER                                           */
/* DESIGN DATE:  JULY 4 ,1984               LAST UPDATE:  September 6,1984  */
/*--------------------------------------------------------------------------*/
/* DESCRIPTION: This file contains a number of small utility modules used   */
/*  by all other modules.                                                   */
/*--------------------------------------------------------------------------*/
/* PSEUDO CODE: All modules are unrelated to each other. The pseudo code of */
/*  each module has been annotated in its own header.                       */
/****************************************************************************/
# include "def.h"
# include "stdio.h"
```

```
/***************************************************************************/
/*                                                                         */
/*                 N O T   _   B L A N K   _   L I N E                     */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                        */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE     */
/*                      FOR INPUT TO STANFORD IC TESTER                    */
/* THESIS ADVISOR:  COL. H.CARTER                                          */
/* DESIGN DATE:  JULY 4 ,1984             LAST UPDATE:  September 6.1984   */
/* SADT REFERENCE:                                                         */
/*-------------------------------------------------------------------------*/
/* DESCRIPTION: This function checks if a given character string is blank  */
/*  or not.                                                                */
/*-------------------------------------------------------------------------*/
/* PSEUDO CODE:                                                            */
/*                  while ( s[i] != '\Ø' )                                 */
/*                    {                                                    */
/*                      if ( alphabetic ( s[i] ))                          */
/*                         {                                               */
/*                           answer = 1;                                   */
/*                           break:                                        */
/*                         }                                               */
/*                     }                                                   */
/*                                                                         */
/*                   return(answer);                                       */
/*-------------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                      */
/*   s: character array, whose status for being 'blank' or otherwise is to */
/*       be determined.                                                    */
/*-------------------------------------------------------------------------*/
/* MODULES CALLED:                                                         */
/*                              alphabetic();                              */
/***************************************************************************/

int not_blank_line(s)
  char  s[]:


 int i=Ø, answer=Ø, true=1;

 while ( s[i] != '\Ø' )
 {
  if ( true )
  {
    if ( alphabetic ( s[i] ))
    {
    answer = 1;
    true   = Ø;
    }
  }
  ++i;
 }
 return(answer);
```

```
/*************************************************************************/
 *                                                                      */
 *                 C O P Y   _   C H A R   _   A R R A Y               */
 *                                                                      */
/*************************************************************************/
 *                                                                      */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                     */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE  */
/*                    FOR INPUT TO STANFORD IC TESTER                    */
/* THESIS ADVISOR:  COL. H.CARTER                                       */
/* DESIGN DATE:  JULY 4 ,1984            LAST UPDATE:  September 6.1984 */
/* SADT REFERENCE:                                                      */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION: This function copies one character array into another.  */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                         */
/*                for ( i=Ø; from[i] != '\Ø'; ++i )                     */
/*                      to[i] = from[i];                                */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                   */
/*   from :  character array from which data is to be copied.           */
/*   to    :  character array into which data is to be copied.          */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                      */
/*                              none                                    */
/*************************************************************************/
copy_char_array (from, to)
 char from[], to[];
{
 int i;
 for ( i=Ø; from[i] != '\Ø'; ++i )
   to[i] = from[i];
 to[i] = '\Ø';
}
```

```
/*************************************************************************/
/*                                                                       */
/*                       C O U N T    W O R D S                          */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                      */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
/*                        FOR INPUT TO STANFORD IC TESTER                */
/* THESIS ADVISOR:  COL. H.CARTER                                        */
/* DESIGN DATE:  JULY 4 ,1984              LAST UPDATE:  September 6.1984 */
/* SADT REFERENCE:                                                       */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION: This function counts the number of words in a given      */
/*   character string.                                                   */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                          */
/*              for ( i = Ø; string [i] != '\Ø' ; ++i )                  */
/*               {                                                       */
/*                if ( alphabetic(string[i]) )                           */
/*                {                                                      */
/*                  if ( looking_for_word )                              */
/*                  {                                                    */
/*                 ++word_count;                                         */
/*                 looking_for_word = Ø;                                 */
/*                  }                                                    */
/*                }                                                      */
/*                else                                                   */
/*                   looking_for_word = 1;                               */
/*               }                                                       */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                    */
/*  string:  character array, in which number of woords is to be counted. */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                       */
/*                                  none                                 */
/*************************************************************************/
int count_words (string)
  char string[];

{
  int i, looking_for_word = 1, word_count = Ø;
  for ( i = Ø; string [i] != '\Ø' ; ++i )
    if ( alphabetic(string[i]) )
    {
      if ( looking_for_word )
      {
        ++word_count;
        looking_for_word = Ø;
      }
    }
    else
       looking_for_word = 1;

  return (word_count );
```

```
/*************************************************************************/
/*                                                                     */
/*                  E Q U A L    -    S T R I N G S                    */
/*                                                                     */
/*************************************************************************/
/*                                                                     */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                    */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
/*                     FOR INPUT TO STANFORD IC TESTER                 */
/* THESIS ADVISOR:  COL. H.CARTER                                      */
/* DESIGN DATE:  JULY    ,1984             LAST UPDATE:                */
/* SADT REFERENCE:                                                     */
/*---------------------------------------------------------------------*/
/* DESCRIPTION:  This function determines if two given character strings */
/*    are equal.                                                       */
/*---------------------------------------------------------------------*/
/* PSEUDO CODE:                                                        */
/*          while ( s1[i] == s2[i] && s1[i] != '\Ø' && s2[i] != '\Ø' )  */
/*                ++i;                                                 */
/*          if ( s1[i] == '\Ø' && s2[i] == '\Ø' )                      */
/*           answer = 1;                                               */
/*          else                                                       */
/*                answer = Ø;                                          */
/*          return(answer);
/*---------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                  */
/*   s1: character array1                                              */
/*   s2: character array2                                              */
/*---------------------------------------------------------------------*/
/* MODULES CALLED:                                                     */
/*                                   none                              */
/*************************************************************************/
int equal_strings( s1, s2 )
  char  s1[], s2[];

{
 int i=Ø, answer;
 while ( s1[i] == s2[i] && s1[i] != '\Ø' && s2[i] != '\Ø' )
  ++i;
 if ( s1[i] == '\Ø' && s2[i] == '\Ø' )
  answer = 1;                                   /*  strings equal  */
 else
  answer = Ø ;                                   /*  not equal      */

 return(answer);
}
```

```
/***************************************************************/
 *                                                             */
 *                 C O P Y     S T R I N G                     */
 *                                                             */
 ***************************************************************/
 *                                                             */
 * STUDENT NAME   :  SQN.LDR.SALEEM                            */
 * THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
 *                     FOR INPUT TO STANFORD IC TESTER         */
 * THESIS ADVISOR:  COL. H.CARTER                              */
 * DESIGN DATE:  JULY 4 ,1984          LAST UPDATE:  September 6.1984 */
 * SADT REFERENCE:                                             */
 *-------------------------------------------------------------*/
 * DESCRIPTION: This function copies a character string from one buffer */
 *   to another buffer.                                        */
 *-------------------------------------------------------------*/
 * PSEUDO CODE:                                                */
 *                for ( ; *from != '\n'; ++from. ++to )        */
 *                    *to = *from;                             */
 *                *to = '\0';                                  */
 *-------------------------------------------------------------*/
 * CALLING ARGUMENTS:                                          */
 *  from : pointer to a character string                       */
 *  to   : pointer to a character string                       */
 *-------------------------------------------------------------*/
 * MODULES CALLED:                                             */
 *                            None                             */
 ***************************************************************/
copy_string (from, to)
  char  *from, *to:

{
   for ( ; *from != '\n'; ++from, ++to )
     *to = *from;

   *to = '\0';
}
```

```
/************************************************************************/
*                                                                      */
*             E X T R A C T     T E S T     D A T A   ( ETD )          */
*                                                                      */
*************************************************************************/
*                                                                      */
* STUDENT NAME   :  SQN.LDR.SALEEM                                     */
* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE  */
*                      FOR INPUT TO STANFORD IC TESTER                 */
* THESIS ADVISOR:  COL. H.CARTER                                       */
* DESIGN DATE:  JULY 4 ,1984            LAST UPDATE:  September 6.1984 */
* SADT REFERENCE:                                                      */
*----------------------------------------------------------------------*/
* DESCRIPTION: This function determines if a given character is alphabetic  */
*----------------------------------------------------------------------*/
* PSEUDO CODE:                                                         */
*     if( (c>='a'&&c<='z')|| (c>='A'&&c<='Z')||(c=='_')||(c>='Ø'&&c<='9') )   */
*         'c' is alphabetic;                                           */
*----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                  */
/* c :  character variable.                                            */
*----------------------------------------------------------------------*/
* MODULES CALLED:                                                      */
/*                                 none                                */
/************************************************************************/
int alphabetic (c)
  char c;
{

 if( (c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c=='_')||(c>='Ø'&&c<='9') )
    return (1);
  else
    return (Ø);
}
```

```
/*************************************************************************/
*                                                                       */
*                   T A B U L A T E   _   C M D  _  D A T A             */
*                                                                       */
*************************************************************************/
*                                                                       */
* STUDENT NAME   :  SQN.LDR.SALEEM                                      */
* THESIS PROJECT:   CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
*                        FOR INPUT TO STANFORD IC TESTER                 */
* THESIS ADVISOR:   COL. H.CARTER                                        */
* DESIGN DATE:  JULY 3 ,1984            LAST UPDATE: November 3, 1984    */
* SADT REFERENCE: A12                                                    */
*-----------------------------------------------------------------------*/
* DESCRIPTION:  This module, on receiving a command line from ESIM file, */
* acts accordingly to set arrays of monitored_pins, input_pins and clock_ */
* pins. It also generates an array of output pins from the available pin */
* data.                                                                  */
*-----------------------------------------------------------------------*/
* PSEUDO CODE:                                                          */
*              if ( a command line )                                    */
*              {                                                        */
*                 if 1st letter of cmd_line(w)                          */
*                   set_up array of monitored_pins.                     */
*                 if 1st letter of cmd_line(V)                          */
*                   set_up array of input_pins.                         */
*                 if 1st letter of cmd_line(K)                          */
*                   set_up array of clock_pins.                         */
*                 if 1st letter of cmd_line(I)                          */
*                   {                                                   */
*                     set correspondence between arrays of input/clock pins */
*                        and elements of monitored_pin array.           */
*                     generate  array of output_pins.                   */
*                   }                                                   */
*                 if 1st letter of cmd_line(R)                          */
*                     ignore.                                           */
*                 if 1st letter of cmd_line(Any other alphabet)         */
*                   activate process 'handle remaining commands'.       */
*              }                                                        */
*-----------------------------------------------------------------------*/
* CALLING ARGUMENTS:                                                    */
*   tptr : pointer to type TABLE, which contains pointers to different  */
*              pin arrays.                                              */
*   fptr : pointer to type PARAMETER, a table which contains all current */
*              program parameters.                                      */
*   cmd_line : A textline read from ESIM file..an array of characters.   */
*   simptr : pointer to array of test vectors generated from node data.  */
*   outfile: pointer to file, in which reformatted data is being stored. */
*-----------------------------------------------------------------------*/
* FILES USED:                                                           */
*   outfile : external file in which reformatted data is stored.        */
*                                                                       */
*-----------------------------------------------------------------------*/
* MODULES CALLED:                                                       */
*                     count_words();                                    */
*                     create_monpin_array();                            */
*                     create_inpin_array();                             */
*                     create_clkpin_array();                            */
*                     generate_outpin_array();                          */
*                     mark_inpin();                                     */
*                     mark_outpin();                                    */
*************************************************************************/
#include "def.h"                  /* file containing all definitions     */
#include"stdio.h"
```

```
int tabulate_cmd_data(tptr, fptr, cmd_line, simptr, outfile)
 TABLE  *tptr;
 PARAMETER   *fptr;
 char cmd_line[];
 VECDATA  *simptr;
 FILE  *outfile;

{
 ICPIN *entry;
 DATA  *simin, *clkin, *outptr;
 char  ch;
 int i,N;
 entry = (tptr)->monpin;                  /* pointer to array of monitored_pins  */
 outptr= (tptr)->outpin;                  /* pointer to array of output pins     */
 simin = (tptr)->inpin;                   /* pointer to array of input pins      */
 clkin = (tptr)->clkpin;                  /* pointer to array of clock pins      */
 ch = cmd_line[Ø];
          switch(ch)
          {
           case 'w':
               if ( cmd_line[1] == ' ' )
                {
                if ( (fptr)->second_time == Ø )
                 {
                  N = count_words(cmd_line);
                  (fptr)->NUM_MONPIN = N-1;
                  create_monpin_array(entry, cmd_line, N-1);
                  (fptr)->second_time = 1;
                 }
                 else
                  {
                  (fptr)->second_time = Ø;
                  (fptr)->pin_chg = 1;
                  }
                }
                  break;

           case 'V':
               if ( cmd_line[1] == ' ' )
                  create_inpin_array(fptr,simin,cmd_line);
                  break;

           case 'K':
               if ( cmd_line[1] == ' ' )
                  create_clkpin_array(fptr,clkin,cmd_line);
                  break;

           case 'I':
                if ( (fptr)->init_again == Ø )
                 {
                  mark_inpin(fptr,entry,simin);
                  mark_clkpin(fptr,entry,clkin);
                  generate_outpin_array(fptr,entry,outptr);
                  (fptr)->init_again = 1;
                 }
                  break;

           case '?':
                  break;

           default :
                if ( cmd_line[1] == ' ' )
                  hand_remcmd(tptr,fptr,cmd_line,simptr,outfile);
                  break;
/***********************************************************************/
/*                                                                     */
/*            C R E A T E    M O N P I N    A R R A Y                  */
/*                                                                     */
/***********************************************************************/
/*                                                                     */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                    */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
/*                       FOR INPUT TO STANFORD IC TESTER               */
/* THESIS ADVISOR:  COL. H.CARTER                                      */
/* DESIGN DATE:  JULY 4 ,1984              LAST UPDATE: November 3, 1984 */
/* SADT REFERENCE: A1221                                               */
/*---------------------------------------------------------------------*/
/* DESCRIPTION: This module, on receiving a command line with command(w) */
/*  sets up an array of monitored pins and stores the number of elements */
/*  in this newly created array in program parameter table.            */
/*---------------------------------------------------------------------*/
/* PSEUDO CODE:                                                        */
/*            f ( a 'w'_command line )                                 */
/*             {                                                       */
/*              for ( i=Ø; i!= number of monpins; ++i)                 */
/*                monpin_array[i].name = get_next_word(stg);           */
/*             }                                                       */
/*---------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                  */
/*  entry : pointer to array of monitored pins.                        */
/*  stg   : a text line read from ESIM file.... an array of characters with */
/*            its first letter as 'w'                                  */
/*  number: an integer, number of elements in array of monitored_pins. */
/*---------------------------------------------------------------------*/
/* MODULES CALLED:                                                     */
/*                  copy_char_array():                                 */
/***********************************************************************/
create_monpin_array( entry,stg, number)
 ICPIN *entry;
 char   stg[];
 int    number;

int i=Ø, j, k=Ø, double_blank=1;
char local[2Ø];

for ( j=2; stg[j] != '\Ø'; ++j )

 if ( alphabetic( stg[j] ))
 {
  local[i] = stg[j]:
  ++i;
  double_blank = Ø;
 }
 else if ( stg[j] == ' ' )
 {
  if ( double_blank == Ø  )
  {
   local[i] = '\Ø';
   copy_char_array(local,(entry+k)->name);
   ++k;
     = Ø;
   double_blank = 1:

  }
 }
 local[ ] = '\Ø';
 copy_char_array(local,(entry+k)->name);
 for (  =Ø; i!= number; ++i )

 {
  entry +  )->class = 'O';
  entr  +  )->class_ref = NOREF;
 }
```

```
/*************************************************************************/
/*                                                                       */
/*            C R E A T E      I N P I N    A R R A Y                    */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* STUDENT NAME  :  SQN.LDR.SALEEM                                       */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
/*                        FOR INPUT TO STANFORD IC TESTER                */
/* THESIS ADVISOR:  COL. H.CARTER                                        */
/* DESIGN DATE:  JULY 8 ,1984                LAST UPDATE: November 3,1984 */
/* SACT REFERENCE: A1223                                                 */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION: This module. on recieving the first commanc line with   */
/*  command(V) sets up an array of input pins and afterwarcs adds  all pin */
/*  name and its associated data in this array. It also stores the number of */
/*  elements in this newly created array in program parameter table.    */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                          */
/*          K = number of elements already stored in inpin_array.(This   */
/*                value initialised to zero at start of program).        */
/*           if ( a 'V'_command line )                                   */
/*             {                                                         */
/*                inpin_array[K+i].name = pin_name (from stg..input parameter) */
/*                inpin_array[K+i].data = pin_data (from stg..input parameter) */
/*             }                                                         */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                    */
/*   fptr : pointer to type PARAMETER, a table which contains all current  */
/*            program parameters.                                        */
/*   simin: pointer to array of input pins.                              */
/*  stg   : a text line read from ESIM file.... an array of characters with */
/*            its first letter as 'V'                                    */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                       */
/*                              None                                     */
/*                                                                       */
/*************************************************************************/
create_inpin_array(fptr,simin,stg)
 PARAMETER  *fptr;
 DATA *simin;
 char stg[]:
{
 int i=0, j=2, K;
 K = (fptr)->in_count;
 while ( stg[j] != ' ' )
 {
  (simin+K)->pin_desig[i] = stg[j]:
  ++i;
  ++j;
 }
 (simin+K)->pin_desig[i] = '\Ø';
 while ( stg[j] == ' ' )
  ++j;

 i = 0;
 while ( stg[j] != '\Ø' )
 {
  (simin+K)->pin_data[i] = stg[j]:
  ++i;
  ++j;
 }
 (simin+K)->pin_data[i] = '\Ø';

 ++((fptr)->in_count);
/*************************************************************************/
*                                                                      */
*           C R E A T E     C L K P I N     A R R A Y                   */
*                                                                      */
/*************************************************************************/
*                                                                      */
* STUDENT NAME   :  SQN.LDR.SALEEM                                     */
* THESIS PROJECT:  CONVERSION OF ECIM DATA FILES TO A FORMAT SUITABLE  */
*                       FOR INPUT TO STANFORD IC TESTER                */
* THESIS ADVISOR:  COL. H.CARTER                                       */
* DESIGN DATE:  JULY 14,1984           LAST UPDATE: November 3, 1984   */
* CADT REFERENCE:  A1222                                               */
*----------------------------------------------------------------------*/
* DESCRIPTION: This module, on recieving the command line with command(K)  */
*  sets up an array of clock pins and adds  all clock_pin names and thier  */
*  associated data in this array. It also stores the number of elements in */
*  this array in the program parameter table.                              */
*----------------------------------------------------------------------*/
* PSEUDO CODE:                                                         */
*             if ( a 'V'_command line )                                */
*             {                                                        */
*               i = 0;                                                 */
*               while ( not eoln )                                     */
*                 {                                                    */
*                   get_next_word;                                     */
*                   clkpin_array[i].name = next_word (from stg..input parameter) */
*                   get_next_word;                                     */
*                   clkpin_array[i].data = next_word (from stg..input parameter) */
*                   ++i;                                               */
*                 }                                                    */
*             }                                                        */
*----------------------------------------------------------------------*/
* CALLING ARGUMENTS:                                                   */
*   fptr : pointer to type PARAMETER, a table which contains all current   */
*               program parameters.                                    */
*   clkpin: pointer to array of clock pins.                            */
*   stg  : a text line read from ECIM file.... an array of characters with */
*               its first letter as 'V'                                */
*----------------------------------------------------------------------*/
* MODULES CALLED:                                                      */
*                               alphabetic();                          */
/*************************************************************************/
create_clkpin_array(fptr,clkpin,stg)
PARAMETER  *fptr;
DATA *clkpin;
char  stg[];

 int i=0, j=2, k;
 k = (fptr->clk_count);

 while ( stg[j] != '\0' )

  while ( stg[j] != ' ' )

    (clkpin+k)->pin_desig[i] = stg[j];
    ++i;
    ++j;

   (clkpin+k)->pin_desig[i] = '\0';

   i = 0;
   ++j;

  while ( alphabetic( stg[j] ) )

    (clkpin+k)->pin_data[i] = stg[j];
    ++i;
    ++j;

  (clkpin+k)->pin_data[i] = '\0';

   ++k;
   if ( stg[j] != '\0' )

    i = 0;
    ++j;
   }
 }
fptr->clk_count = k;
```

```
/************************************************************************/
*                                                                      */
*            G E N E R A T E   O U T P I N   A R R A Y                 */
*                                                                      */
/************************************************************************/
*                                                                      */
* STUDENT NAME  :  SQN.LDR SALEEM                                      */
* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE  */
*                  FOR INPUT TO STANFORD IC TESTER                     */
* THESIS ADVISOR:  COL. H.CARTER                                       */
* DESIGN DATE:  JULY 20,1984           LAST UPDATE: October 29,1984    */
* LAST REFERENCE:  A1024                                               */
*----------------------------------------------------------------------*/
* DESCRIPTION:  This module scans the array of monitored pins and sets up  */
*  a separate array called outpin_array consisting of pins not marked as   */
*  input or clock pins.                                                */
*                                                                      */
*----------------------------------------------------------------------*/
* PSEUDO CODE:                                                         */
*             int j=0;                                                 */
*             for ( i=0; i!= number of elements in monpin array; ++i ) */
*                {                                                     */
*                 if ( element[i].class != input or clock )            */
*                   {                                                  */
*                    outpin_array[j] = monpin_element[i];              */
*                    ++j;                                              */
*                   }                                                  */
*                }                                                     */
*----------------------------------------------------------------------*/
* CALLING ARGUMENTS:                                                   */
*   fptr : pointer to type PARAMETER, a table which contains all current  */
*           program parameters.                                        */
*  entry : pointer to array of monitored pins.                         */
*  outptr: pointer to array of output pins.                            */
*----------------------------------------------------------------------*/
* MODULES CALLED:                                                      */
*                   copy_char_array():                                 */
/************************************************************************/
generate_outpin_array(fptr,entry,outptr)
 PARAMETER  *fptr;
 ICPIN *entry;
 DATA  *outptr;

 {
 int i, J, K;
 J = (fptr)->NUM_MONPIN;
 K = (fptr)->out_count;
 for ( i=0; i!= J; ++i )
   {
   if ( (entry+i)->class == 'O' )
     {
     copy_char_array( (entry+i)->name, (outptr+K)->pin_desig);
     (entry +i)->class_ref = K;
     (outptr + K)-> monref = i;
     ++K;
     }
   }
  fptr->out_count = K;
  /* DEBUG */

  printf("\n output: generate outpin array function \n");
  for ( i=0; i!= fptr->out_count; ++i)
   {
   printf("%3s \n[%d]",i,(outptr+i)->pin_desig);

   printf("%d \n", (outptr+i)->monref);
```

```
/***********************************************************************/
/*                                                                     */
/*           C H A N G E _ S T A T U S _ H L C M D                     */
/*                                                                     */
/***********************************************************************/
/*                                                                     */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                    */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
/*                     FOR INPUT TO STANFORD IC TESTER                 */
/* THESIS ADVISOR:  COL. H.CARTER                                      */
/* DESIGN DATE:  Sept 25,1984              LAST UPDATE: October 28,1984 */
/* SADT REFERENCE: A1232     ...... presently not implemented......... */
/*---------------------------------------------------------------------*/
/* DESCRIPTION:  This module on recieving a valid 'hl-cmd' restructures the */
/*  previous available node data into test vectors, stores these vectors in */
/*  an external file and changes the status of effected pins by filling in  */
/*  their respective buffers with 'Ø or 1' depending on ' l or h' command   */
/*  respectively.                                                      */
/*---------------------------------------------------------------------*/
/* PSEUDO CODE:                                                        */
/*              set stat = Ø     if 'l_cmd'                            */
/*              set stat = 1     if 'h_cmd'                            */
/*              for ( i=Ø; i!= number of elements in hl_array: ++i)    */
/*              {                                                      */
/*                for ( j=Ø; j!= total number of input pins; ++j )     */
/*                {                                                    */
/*                  if ( hl_array_element[i] = inpin_array_element[j]) */
/*                  {                                                  */
/*                    fill in respective buffer of inpin_array_element[j] */
/*                      with 'stat'.                                   */
/*                    delete crossref with array of monitored pins.    */
/*                  }                                                  */
/*                  else                                               */
/*                  {                                                  */
/*                    add new element to inpin array ( hl_array_element[i]) */
/*                    fill in respective buffer of new_element with 'stat' */
/*                  }                                                  */
/*                }                                                    */
/*              }                                                      */
/*              in_count = previous in_count + number of new elements added */
/*---------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                  */
/*    tptr : pointer to type TABLE, which contains pointers to different */
/*              pin arrays.                                            */
/*    fptr : pointer to type PARAMETER, a table which contains all current */
/*              program parameters.                                    */
/*    hlarray : pointer to an array set up in calling routine to store */
/*                elements recieved in 'h or l' command line.          */
/*    hlcount : integer, giving number of words in hl_cmd_line         */
/*    stat    : integer flag for h or l command line.                  */
/*    simptr : pointer to array of test vectors generated from node data. */
/*    outfile: pointer to file, in which reformatted data is being stored. */
/*---------------------------------------------------------------------*/
/* MODULES CALLED:                                                     */
/*                presently not implemented();                         */
/***********************************************************************/
```

```
/**********************************************************************/
 *                                                                   */
 *                 V A L I D A T E _ C M D L I N E                   */
 *                                                                   */
/*********************************************************************/
/*                                                                   */
/* STUDENT NAME  :  SQN.LDR.SALEEM                                   */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
/*                     FOR INPUT TO STANFORD IC TESTER               */
 * THESIS ADVISOR:  COL. H.CARTER                                    */
 * DESIGN DATE:  Sept 6 ,1984              LAST UPDATE: Octcber 25, 1984 */
 * SADT REFERENCE:  A1231                                            */
 *-------------------------------------------------------------------*/
 * DESCRIPTION:  This module checks if the given text line is either 'h' */
 *  or 'l' command line. It ignores all other textlines recieved from */
 *  ESIM file. It perticularly checks against another line format, which */
 *  although starts with 'h/l' but actualy describes different nodes in */
 *  high or low status.                                              */
 *-------------------------------------------------------------------*/
 * PSEUDO CODE:                                                      */
 *              if ( first element of char.string is 'h or l')       */
 *                line = valid line for checking further.            */
 *               if ( valid line )                                   */
 *                 {                                                 */
/*                  X = strncmp( line , "h inputs:" ):               */
/*                  Y = strncmp( line , "h inputs:" ):               */
/*                  if ( X == Ø || Y == Ø )                          */
/*                      line = invalid                               */
/*                 }                                                 */
 *               return ( line_validity )                            */
/*-------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                */
/*  stg : character array                                            */
/*-------------------------------------------------------------------*/
/* MODULES CALLED:                                                   */
/*                       None                                        */
/*********************************************************************/
int validate_cmdline(stg)
 char stg[]:

{
 int x,y,R=Ø:
 char *hptr, *lptr:

 hptr = "h inputs:":
 lptr = "l inputs:":

 if ( stg[Ø] == 'h' || stg[Ø] == 'l' )
  {
    x = strncmp( stg, hptr, 8 ):
    y = strncmp( stg, lptr, 8 ):
    if ( x == Ø || y == Ø )
      R = -1:
  }
 else
  R = -1:

  if ( R == -1 )
    printf(" invalid hl line : %s\n",stg):
  else
    printf(" valid hl line : %s\n",stg):

  return(R
/*************************************************************************/
/*                                                                       */
/*                      S C R U T   _   H L C M D                        */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* STUDENT NAME  :  SQN.LDR.SALEEM                                       */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
/*                      FOR INPUT TO STANFORD IC TESTER                  */
/* THESIS ADVISOR:  COL. H.CARTER                                        */
/* DESIGN DATE:  Sept 3 ,1984          LAST UPDATE:  October 27, 1984    */
/* SADT REFERENCE:  A12313                                               */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION:  This module validates the 'h or l' command line by checking */
/*  that no element of 'hl_cmd_line' has already been designated as output  */
/*  pin. It generates an error message and aborts the program.           */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                          */
 *               for ( i=Ø;  i!= number of elements in hl_array; ++i)    */
 *                {                                                      */
 *                 for ( j=Ø: j!= total number of output pins;  ++j )    */
/*                  {                                                    */
/*                   if ( hl_array_element[i] == outpin_array_element[j] */
/*                    {                                                  */
/*                     hl_command = invalid;                             */
/*                     break;                                            */
/*                    }                                                  */
/*                  }                                                    */
/*                }                                                      */
/*               return( hl_cmd_status )                                 */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                    */
/*   outptr  : pointer to array of output pins.                          */
/*   hlarray : pointer to an array set up in calling routine to store    */
/*               elements recieved in 'h or l' command line.             */
/*   outcount: integer, giving number of elements in array of output pins */
/*   hlcount : integer, giving number of words in hl cmd_line            */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                       */
/*                        strncmp();                                     */
/*************************************************************************/
scrut_hlcmd( outptr, hlarray, outcount, hlcount )
 DATA   *outptr, *hlarray;
 int    outcount, hlcount;

{
 int i,j, found = Ø;

 for ( i=Ø; i!= hlcount; ++i )
  {
   for ( j=Ø; j!= outcount; ++j)
    {
     if(strncmp((hlarray+i)->pin_desig,(outptr+j)->pin_desig,2Ø)==Ø)
      {
       printf("INVALID PIN : %s",(hlarray+i)->pin_desig);
       printf(" is in the list of 'output' pins.\n");
       found = 1;
       break;
      }
     if ( found )
      break;
    }
  }
  return(found);
}
```

```
/*************************************************************************/
/*                                                                       */
/*              M A K E      H L   C M D     A R R A Y                   */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                      */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
/*                      FOR INPUT TO STANFORD IC TESTER                  */
/* THESIS ADVISOR:  COL. H.CARTER                                        */
/* DESIGN DATE:  Sept 8 ,1984            LAST UPDATE:  October 26, 1984  */
/* SADT REFERENCE:  A12311                                               */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION: This module sets up an array of pins included in 'h'     */
/*  or 'l' command line... whose status is going to be charged to 'high' */
/*  or 'low'                                                             */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                          */
/*              for ( i=Ø; i!= number of words in hlcmd_line; ++i)       */
/*                {                                                      */
/*                 get_next_word;                                        */
/*                 hlcmd_array_element[i] = next_word;                   */
/*                }                                                      */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                    */
/*   hlarray : pointer to an array set up in calling routine to store   */
/*               elements recieved in 'h or l' command line.             */
/*   textline: a character array...a command line read from ESIM file.  */
/*   hlcount : integer, giving number of words in hlcmd_line             */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                       */
/*                     None                                              */
/*                                                                       */
/*************************************************************************/
 make_hlcmd_array( hlarray, textline, hlcount)
  DATA  *hlarray;
  char  textline[];
  int   hlcount;

 {
  int i, j, k, l, blank2=1;
  k=l=Ø;

  for ( j=1; textline[j]!= '\Ø'; ++j )
   {
    if ( alphabetic(textline[j]) )
     {
      (hlarray+k)->pin_desig[l] = textline[j];
      +-l;
      blank2 = Ø;
     }

    if ( textline[j] == ' ' && blank2 == Ø )
     {
      (hlarray+k)->pin_desig[l] = '\Ø';
      ++k;
      l = Ø;
      blank2 = 1;
     }
   }
  (hlarray+k)->pin_desig[l] = '\Ø';
  for ( i =Ø; i != hlcount; ++i )
   (hlarray + i)->morref = 99;
 }
```

```
int hand_remcmd(tptr,fptr,stg,simptr,outfile)
 TABLE   *tptr;
 PARAMETER   *fptr;
 char    stg[];
 VECDATA   *simptr;
 FILE    *outfile;

 {
  int i,j,K,L,M,P,Q,valid, err=Ø;
  ICPIN    *entry;
  DATA     *inptr, *outptr, *hlptr;

  outptr= (tptr)->outpin;
  hlptr= (tptr)->hlnode;
  K = (fptr)->NUM_MCNPIN;
  L = (fptr)->in_count;
  M = (fptr)->out_ccunt;

  P = count_words(stg) - 1;
  valid = validate_cmdline( stg );
  if ( valid == Ø)
  {
   printf("is a valid line: %s\n", stg);
   make_hlcmd_array( hlptr, stg, P );

    for( i=Ø; i!= P; ++i)
     printf(" %5d   %s\n",i,(hlptr+i)->pin_desig);

    if ( stg[Ø] == 'h')
      Q = 1;
    else
      Q = Ø;

    if ( scrut_hlcmd(outptr, hlptr, M, P) )
      err = -1;
   }

  return(err);
}
```

```
/****************************************************************************/
/*                                                                          */
/*                       HANDLE  REMAINING COMMANDS                         */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                         */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE      */
/*                     FOR INPUT TO STANFORD IC TESTER                      */
/* THESIS ADVISOR:  COL. H.CARTER                                           */
/* DESIGN DATE:  Sept 2 ,1984             LAST UPDATE: October 25, 1984     */
/* SADT REFERENCE:  A123                                                    */
/*--------------------------------------------------------------------------*/
/* DESCRIPTION: This module on recieving any other command except( w,V,K,I  */
/*  and R ) ignores all others...presently...and processes 'h or l' cmd     */
/*  only. It checks the validity of the command and changes the status of   */
/*  effected pins.                                                          */
/*                                                                          */
/*--------------------------------------------------------------------------*/
/* PSEUDO CODE:                                                             */
/*               if ( a h_cmd or a l_cmd )                                  */
/*                 {                                                        */
/*                    change node data into vector form:                    */
/*                    write out test vectors to external file:              */
/*                    make array of pins included in 'h or l' command line; */
/*                    scrutinize hl-commands for its validity:              */
/*                    change status of effected pins;                       */
/*                 }                                                        */
/*--------------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                       */
/*   tptr : pointer to type TABLE, which contains pointers to different     */
/*          pin arrays.                                                     */
/*   fptr : pointer to type PARAMETER, a table which contains all current   */
/*          program parameters.                                             */
/*   stg  : A textline read from ESIM file..an array of characters.. an     */
/*          h or l command line                                             */
/*   simptr : pointer to array of test vectors generated from node data.    */
/*   outfile: pointer to file, in which reformatted data is being stored.   */
/*--------------------------------------------------------------------------*/
/* FILES USED:                                                              */
/*   outfile : external file in which reformatted test vectors are stored.  */
/*                                                                          */
/*--------------------------------------------------------------------------*/
/* MODULES CALLED:                                                          */
/*                    make_hlcmd_array();                                   */
/*                    scrut_hlcmd();                                        */
/*                    change_status_hlcmd();    ..... presently not implemented. */
/****************************************************************************/
#include "stdio.h"
#include "cef.h"
```

```
        Q = (inptr+j)->monref;
        (simptr+i)->test_input[j] = (entry+Q)->datastg[i];
       }
     else
        (simptr+i)->test_input[j] = (inptr+j)->pin_data[i];
    }

   for ( j=Ø; j!=M; ++j )
    {
     Q = (outptr+j)->monref;
     (simptr+i)->ref_output[j] = (entry+Q)->datastg[i];
    }

   (simptr+i)->follow = 1;
   (simptr+i)->change = Ø;
  }

   (fptr)->prn_count = N;
  if ( (fptr)->final == 1 )
    (simptr-N-1)->follow = Ø;

}
```

```
/*************************************************************************/
/*                                                                       */
/*              C O N V E R T _ N O D E 2 _ V E C T O R                  */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                      */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
/*                     FOR INPUT TO STANFORD IC TESTER                   */
/* THESIS ADVISOR:  COL. H.CARTER                                        */
/* DESIGN DATE: August 28. 1984                                          */
/* DESIGN DATE:     August 28, 1984          LAST UPDATE: October 2Ø,1984 */
/* SADT REFERENCE:  A1322                                                */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION:  This module scans the input array and forming an array  */
/*   consisting of data string associated with each input pin....each data */
/*   string is itself an array of characters,it transposes the imaginary */
/*   matrix to convert node data into test vectors.                      */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                          */
/*       M = number of elements in array of input pins                   */
/*       N = size of data string associated with inpin_array_element[Ø] */
/*       form an imaginary matrix of size ( M x N ) with its rows as     */
/*          data strings.                                                */
/*       Take a transpose and resulting in a matrix of size ( N x M )    */
/*          where  N.  is now number of test vectors and M is size of    */
/*          each test vector.                                            */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                    */
/*   tptr : pointer to type TABLE, which contains pointers to different  */
/*           pin arrays.                                                 */
/*   fptr : pointer to type PARAMETER, a table which contains all current */
/*           program parameters.                                         */
/*   simptr : pointer to array of test vectors generated from node data. */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                       */
/*                         None                                          */
/*************************************************************************/
convert_node2_vector(tptr,fptr,simptr)
  TABLE    *tptr:
  PARAMETER  *fptr:
  VECDATA    *simptr:

  {
   int i,j,K,L,M,N,P,Q:
   ICPIN    *entry:
   DATA     *inptr, *outptr;
   char     *temp:

   entry = (tptr)->monpin:
   inptr = (tptr)->inpin;
   outptr= (tptr)->outpin:

   K = (fptr)->NUM_MCNPIN:
   L = (fptr)->ir_count:
   M = (fptr)->out_ccunt:
   N = strlen( (entry)->datastg):

 for ( i=Ø: i!=N: ++i )
  {
   for (  j=Ø: j!=L: ++j )
     {
      if ( (inptr+j)->monref != 99 )
/******************************************************************************/
/*                                                                            */
/*                C H A N G E  _  D A T A  _  S T R U C T U R E               */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                           */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE        */
/*                     FOR INPUT TO STANFORD IC TESTER                        */
/* THESIS ADVISOR:  COL. H.CARTER                                             */
/* DESIGN DATE:  August 18, 1984            LAST UPDATE: October 18, 1984     */
/* SADT REFERENCE: A132                                                       */
/*----------------------------------------------------------------------------*/
/* DESCRIPTION:  This module singles out the input pins not included in the  */
/*   array of monitored pins and extrapolates the data associated with these */
/*   pins to match with other monitored pins. It then changes the pin node    */
/*   data into test vectors by calling 'reformat_node_data' routine.          */
/*                                                                            */
/*----------------------------------------------------------------------------*/
/* PSEUDO CODE:                                                               */
/*             for ( i=Ø; i!= number of input pins; ++i)                      */
/*               {                                                            */
/*                 if ( inpin_array_element[i] not a member of monpin array ) */
/*                   {                                                        */
/*                     N = size of associated data string                     */
/*                     fill ( N to buffer_size) places with (N-1)th place data */
/*                   }                                                        */
/*               }                                                            */
/*             change node data into vector form ( convert_node2_vector)      */
/*----------------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                         */
/*   tptr : pointer to type TABLE, which contains pointers to different       */
/*              pin arrays.                                                   */
/*   fptr : pointer to type PARAMETER, a table which contains all current     */
/*              program parameters.                                           */
/*   simptr : pointer to array of test vectors generated from node data.      */
/*----------------------------------------------------------------------------*/
/* MODULES CALLED:                                                            */
/*                        convert_noce2_vector();                             */
/******************************************************************************/
change_data_structure(tptr,fptr, simptr)
 TABLE  *tptr;
 PARAMETER *fptr;
 VECDATA  *simptr;

{
  int i,j,K,L,M,N,P,Q;
  ICPIN *entry;
  DATA  *inptr, *outptr;
  char  *makenull;
  char  trial[2Ø];
  entry = (tptr)->monpin;
  inptr = (tptr)->inpin;
  outptr= (tptr)->outpin;
  K = (fptr)->NUM_MONPIN;
  L = (fptr)->in_count;
  M = (fptr)->out_count;
  N = strlen ( (entry)->datastg );
  makenull = "";

 for (i=Ø;  i!= L; ++i )

     if (( inptr+i)->monref == 99 )


       P = strlen( (inptr+i)->pin_data );
       for ( j=P-1; j!=N; ++j)
        (inptr+i)->pin_data[j] = (inptr+i)->pin_data[P-1];
        (inptr+i)->pin_data[N] = '\Ø';
       }
   }
  convert_node2_vector(tptr,fptr,simptr);

 for ( i=Ø; i!= K; ++i)
  strcpy ( (entry+i)->datastg, makenull );
}
```

```
 }
 dstg[i] = '\Ø';
 ++j;

 i= Ø;
 while ( dataline[j] != '\Ø' )
 {
  pinstg[i] = dataline[j];
  ++i;
  ++j;
 }
 pinstg[i] = '\Ø';

  K = (fptr)->dstg_count;
  M = strlen( (entry+K)->datastg );
  N = strlen(dptr);

 if ( (K==Ø) && (M+N)>MAXL )
  (fptr)->overflow = 1;
 else
  {
  strncat( (entry+K)->datastg, dptr,4Ø);
  ++( (fptr)->dstg_count);
  if ( (fptr)->dstg_count == (fptr)->NUM_MONPIN )
   (fptr)->dstg_count = Ø;
  strcpy( dataline, makenull);
  }
}
```

```
/**************************************************************************/
/*                                                                        */
/*              S T O R E   -   N O D E   -   D A T A                     */
/*                                                                        */
/**************************************************************************/
/*                                                                        */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                        */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE    */
/*                     FOR INPUT TO STANFORD IC TESTER                    */
/* THESIS ADVISOR:  COL. H.CARTER                                          */
/* DESIGN DATE:  August 2 ,1984            LAST UPDATE: October 27, 1984  */
/* SADT REFERENCE:  A131                                                   */
/*------------------------------------------------------------------------*/
/* DESCRIPTION:  This module, on recieving a data_line from ESIM file     */
/*  segregates it into its name and data parts. It then adds the node data */
/*  into its respective buffer if enough space is available; otherwise an  */
/*  overflow flag is generated to clear the buffers before adding any new  */
/*  data.                                                                  */
/*------------------------------------------------------------------------*/
/* PSEUDO CODE:                                                            */
/*               K = repitition of data line in monpin group;             */
/*               dstring = data part of data line;                         */
/*               if ( (size of dstring + data in Kth buffer) > buffer size ) */
/*                  set buffer overflow flag.                              */
/*               else                                                      */
/*                  {                                                      */
/*                   add data to Kth buffer.                               */
/*                   --K;                                                  */
/*                   if (  K == number of monitored pins )                 */
/*                    set K = Ø; ( restart next group )                    */
/*                  }                                                      */
/*------------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                      */
/*    tptr : pointer to type TABLE, which contains pointers to different   */
/*             pin arrays.                                                 */
/*    fptr : pointer to type PARAMETER, a table which contains all current */
/*             program parameters.                                         */
/*    data_line : A textline read from ESIM file..an array of characters.  */
/*            first letter being '>'.                                      */
/*------------------------------------------------------------------------*/
/* MODULES CALLED:                                                         */
/*                         None                                            */
/**************************************************************************/
 store_node_data(tptr,fptr,dataline)
 TABLE  *tptr;
 PARAMETER   *fptr;
 char  dataline[];

{
  int i=Ø, j=1, L, K, M, N;
  char *pinptr, *dptr, *makenull;
  char pinstg[2Ø], dstg[1ØØ];
  ICPIN  *entry;

  pinptr = pinstg;
  dptr    = dstg;
  entry = (tptr)->monpin;
  makenull = "";

  while ( dataline[j] != ':' )
   {
     dstg[i] = dataline[j];
     ++i;
     ++j;
```

```
/*************************************************************************/
/*                                                                       */
/*              R E S T R U C T U R E   T E S T   D A T A                */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                      */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE   */
/*                     FOR INPUT TO STANFORD IC TESTER                   */
/* THESIS ADVISOR:  COL. H.CARTER                                        */
/* DESIGN DATE:  August 2, 1984          LAST UPDATE: October 27, 1984   */
/* SADT REFERENCE: A13                                                   */
/*-----------------------------------------------------------------------*/
/* DESCRIPTION: This module stores node data into its respective buffer thru */
/*  a sub_routine, changes its format into test vectors and stores them in   */
/*  a given external file.                                               */
/*-----------------------------------------------------------------------*/
/* PSEUDO CODE:                                                          */
/*              if ( textline is a 'cmd_line')                           */
/*                store_node_data();                                     */
/*              if ( data overflow == 1 )                                */
/*                {                                                      */
/*                 change_data_structure();                              */
/*                 append2_file();                                       */
/*                 store_node_data();                                    */
/*                }                                                      */
/*-----------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                    */
/*   tptr : pointer to type TABLE, which contains pointers to different  */
/*              pin arrays.                                              */
/*   fptr : pointer to type PARAMETER, a table which contains all current */
/*              program parameters.                                      */
/*   data_line : A textline read from ESIM file..an array of characters. */
/*             first letter being '>'.                                   */
/*   simptr:pointer to array of test vectors generated from node data.   */
/*   outfile: pointer to file in which test vectors are to be stored.    */
/*-----------------------------------------------------------------------*/
/* MODULES CALLED:                                                       */
/*                     store_node_data();                                */
/*                     change_data_structure();                          */
/*                     append2_file();                                   */
/*************************************************************************/
#include"def.h"
#include"stdio.h"

restructure_testdata(tptr, fptr, data_line, simptr, outfile)
 TABLE   *tptr;
 PARAMETER   *fptr;
 char    data_line[];
 VECDATA     *simptr;
 FILE    *outfile;

{
  if ( (fptr)->final != Ø )
     store_node_data( tptr, fptr, data_line);
  if ( (fptr)->overflow == 1 || (fptr)->final == 1)
    {
      change_data_structure( tptr, fptr, simptr);
      append2_file( fptr, tptr, simptr, outfile);
    }
  if ( (fptr)->overflow == 1 )
    {

      (fptr)->overflow = Ø;
      store_node_data( tptr, fptr, data_line);
    }
```

```
/***********************************************************************/
 *                                                                    */
/*                    M A R K          C L K P I N                     */
 *                                                                    */
/***********************************************************************/
/*                                                                     */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                    */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
/*                     FOR INPUT TO STANFORD IC TESTER                 */
/* THESIS ADVISOR:  COL. H.CARTER                                      */
/* DESIGN DATE:  JULY 13,1984            LAST UPDATE: November 3, 1984 */
/* SADT REFERENCE:  A12242                                             */
/*---------------------------------------------------------------------*/
/* DESCRIPTION:  This module compares each element of array of clock pins */
/*  with all elements of array of monitored pins and sets a correspondence */
/*  between the elements of borh arrays when a match is found.         */
/*                                                                     */
/*---------------------------------------------------------------------*/
/* PSEUDO CODE:                                                        */
/*                  for( i=Ø; i!= total number of clock pins: ++i)     */
/*                    {                                                */
/*                     for ( j=Ø; j!= total number of monitored pins:  ++j) */
/*                      {                                              */
/*                        if(clkpin_array_element[i]==monpin_array_element[j] ) */
/*                         {                                           */
/*                           monpin_array_element[j].class = 'CLOCK':  */
/*                           monpin_array_element[j].crsref=  i;       */
/*                           clkpin_array_element[i].monref=  j;       */
/*                         }                                           */
/*                      }                                              */
/*                    }                                                */
/*---------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                  */
/*   fptr : pointer to type PARAMETER, a table which contains all current */
/*          program parameters.                                        */
/*   clkx : pointer to array of monitored pins.                        */
/*   clksim:pointer to array of input pins.                            */
/*---------------------------------------------------------------------*/
/* MODULES CALLED:                                                     */
/*                   equal_strings();                                  */
/*                                                                     */
/***********************************************************************/
mark_clkpin(fptr,clkx,clksin)
 PARAMETER   *fptr;
 ICPIN *clkx:
 DATA   *clksim:

{
 int i, j:

 for ( i=Ø; i != (fptr)->clk_count: ++i )
 {
   for ( j=Ø; j!= (fptr)->NUM_MONPIN: ++j )
     {
     if(equal_strings((clksim+i)->pin_desig,(clkx+j)->name))
      {
       (clkx + j)->class =  'C';
       (clkx + j)->class_ref = i:
       (clksim+i)->monref = j;
       break;
      }
     else
       (clksim + i)->monref = NOREF:
     }

 }
 if ( DEBUG )
   {
    printf("executing create_clkpin_array \n"):
    for ( i=Ø; i!= (fptr)->clk_count; ++i)
     {
      printf(" %c", i :
      printf(" %s",(clksim +i)->pin_desig);
      printf("  %d",(clksim +i)->monref):
      printf(" %s\n",(clksim +i)->pin_data):
     }
   }
```

```
/***********************************************************************/
/*                                                                     */
/*                      M A R K           I N P I N                    */
/*                                                                     */
/***********************************************************************/
/*                                                                     */
/* STUDENT NAME   :  SQN.LDR.SALEEM                                    */
/* THESIS PROJECT:  CONVERSION OF ESIM DATA FILES TO A FORMAT SUITABLE */
/*                      FOR INPUT TO STANFORD IC TESTER                */
/* THESIS ADVISOR:  COL. H.CARTER                                      */
/* DESIGN DATE:  JULY 14,1984            LAST UPDATE: November 3, 1984 */
/* SADT REFERENCE: 12241                                               */
/*---------------------------------------------------------------------*/
/* DESCRIPTION:  This module compares each element of array of input pins */
/*  with all elements of array of monitored pins and sets a correspondence */
/*  between the elements of borh arrays when a match is found.        */
/*                                                                     */
/*---------------------------------------------------------------------*/
/* PSEUDO CODE:                                                        */
/*              for( i=Ø; i!= total number of input pins; ++i)         */
/*                {                                                    */
/*                 for ( j=Ø; j!= total number of monitored pins;  ++j) */
/*                  {                                                  */
/*                    if( inpin_array_element[i]==monpin_array_element[j] ) */
/*                     {                                               */
/*                       monpin_array_element[j].class = 'INPUT';      */
/*                       monpin_array_element[j].crsref=  i;           */
/*                        inpin_array_element[i].monref=  j;           */
/*                     }                                               */
/*                  }                                                  */
/*                }                                                    */
/*---------------------------------------------------------------------*/
/* CALLING ARGUMENTS:                                                  */
/*    fptr : pointer to type PARAMETER, a table which contains all current */
/*           program parameters.                                       */
/*    entry: pointer to array of monitored pins.                       */
/*    simin: pointer to array of input pins.                           */
/*---------------------------------------------------------------------*/
/* MODULES CALLED:                                                     */
/*                    equal_strings();                                 */
/***********************************************************************/
mark_inpin(fptr,entry,simin)
 PARAMETER  *fptr;
 ICPIN     *entry;
 DATA      *simin;

 {
 int i,j,K;
 K = (fptr)->in_count;
 for ( i=Ø; i!= k; --i )
  {
   (simin + i)->monref = NOREF;
   for ( j=Ø; j != (fptr)->NUM_MONPIN; ++j )
   {
    if ( equal_strings( (entry+j)->name, (simin+i)->pin_desig ))
     {
      (entry + j)->class = 'I';
      (entry + j)->class_ref = i;
      (simin + i)->monref = j;
       break;
     }
   }
  }

  if ( DEBUG )
  {
   printf("executing mark_inpin: \n");
   for ( i=Ø; i != k; ++i)
    {
     printf("%d    %s",i,(simin+i)->pin_desig);
     printf(" %d  %s\n",(simin+i)->monref,(simin+i)->pin_data);
    }
  }
}
```

Appendix "E"

SIEVE

A Functional Test Specification

Interchange Format

Introduction

The enclosed format provides for the specification of a procedure for functionally testing digital microchips.

You may specify a sequence of input vectors, binary values for your chip's input pins, and also a sequence of output vectors, binary values expected of your chip's output pins.

This interchange format ultimately drives testing equipment so that your chip receives your specified input vectors and has its outputs checked against your specified output vectors. Any time your chip's ouputs do not match your specified output vectors, the test procedure notifies you of the clock cycle and the pin with the unexpected output value.

Note: This interchange format is to be read and written primarily by machines and not people. Therefore, feel free to write your own test language and programs to translate your favorite language into this particular format. (The introduction of macros in this format was motivated not by human convenience, but for the potential of saving disk space).

Getting Started

The interchange format consists of two parts, a "declarations" section and a "body" section.

The declaration section gives names to your chip's pins so that you can specify subsequently, in the "body" section, your test vectors in terms of these names for pins, as opposed to specifying something horrible such as geometric coordinates for your pins each time you want to reference each pin.

It is also in the declaration section where you specify which pins are to receive power (VDD) and ground (GND).

Finally, the declaration section provides a rudimentary, parameterless "macro" definition capability. That is, you may define abbreviations in the declaration section which will be understood within the "body" section. This capability provides for shorter interchange files.

Comments may be placed anwhere, and have the format:

```
/*  arbitrary text  */
```

In addition, blanks (spaces, tabs, carriage-returns, line-feeds, form-feeds) may appear anywhere.

The Declaration Section

The first thing to appear in your interchange format file is the "declaration" section. There are five parts in this section, and they may appear in any order.

Each part begins with a keyword followed by a colon.

Four of the five parts serve to assign numbers to pins. The remaining part provides for macro definitions.

Providing Names (Numbers) For Your Pads/Pins

Imagine your chip as a box, which has of course four sides, TOP, BOTTOM, LEFT, and RIGHT. You assign numbers to pins separately for each of the four sides:

```
TOP:        # , # ,  ...  , # ;
BOTTOM:     # , # ,  ...  , # ;
RIGHT:      # , # ,  ...  , # ;
LEFT:       # , # ,  ...  , # ;
```

[Nomenclature: We will use the sharp sign (#) to designate any string of digits.]

The order of the numbers following each of the four edge-keywords is important. We assume that you assign numbers along the TOP and BOTTOM edges left-to-right. We assume also that you assign numbers along the LEFT and RIGHT edges bottom-to-top. In other words, we expect always that the order corresponds to increasing-X or increasing-Y coordinates.

We reserve the numbers 0 and 1, which designate that the corresponding pin is to be tied to GND or VDD forever!

We wish to stress that this interchange format is meant to be generic. We intend it to be sufficiently non-specific so as to apply either to pads on a chip or to pins on a package, etc. For a particular application, you will need in addition to this document another document which dictates exactly how many elements are expected along the TOP, BOTTOM, LEFT, and RIGHT.

For example, if we are testing a packaged chip, TOP, BOTTOM, LEFT, and RIGHT correspond to the four sides of the package itself. (A 40-pin DIP will expect 20 entries in each of TOP and BOTTOM, and zero entries in each of LEFT and RIGHT).

For another example, if we are testing via direct wafer probe, we will generally insist that your project employs some well-defined "standard pad frame" (for which we will have constructed a "standard probe card"). In this case once again, the lengths of each of TOP, BOTTOM, LEFT, and RIGHT are fixed, and so your positional association of signal numbers along each edge continues to be well-defined.

Macros

The fifth part of the declarations sections provides for macro definitions:

```
MACRO:  name { arbitrary text }
MACRO:  name { arbitrary +ext }
  . . .
MACRO:  name { arbitrary text }
```

Each macro definition is preceded by the keyword MACRO followed by a colon. Then comes the macro name (the abbreviation which you will use later in the "body"). The expanded meaning of the abbreviation appears following the name, enclosed in curly brackets ({}).

We restrict macro names to consist only or letters and digits and underscores (_). We will make distinction between upper and lower case letters. We insist also that macro names be unique in their first 64 characters, (e.g., you're safe if all your macro names contain no more than 64 characters).

Finally, the order of macro definition is unrestricted. In fact, MACROs may be "nested", that is, a macro may call another macro from with its definition (the "{arbitrary test}" part).

The Body: Your Test Vectors

The body starts off with the keyword

BEGIN_TEST:

and continues with a sequence of any of four directives, and finally terminates with the keyword

END_TEST

The four kinds of directives are:

"Drive a pin to a specified value"

"Verify that a pin presently has a specified value"

"Complete specifications received thus far before proceeding"

"Call a macro"

Let's discuss the first two, most popular, directives first. You actually specify not an input vector at a time, but rather one element of an input vector at a time. You actually say "drive this single pin to this single value".

You specify an entire vector by specifying many "drive pin" directives.

The "drive pin" directive appears as follows:

= # ;   that is, pin = value ;

The semicolon is part of the specification. The first number designates the pin and the second number specifies the value, 0 or 1. (Please remember that this pin number is unrelated to any numbering of pins that might be associated with a chip package, rather, this pin number is associated to your chip via "The Declaration Section").

The "verify pin" directive has a similar format, where the "=" is replaced by "?":

? # ;   that is, pin ? value;

Again, the semicolon is part of the specification.

Framing

The third kind of directive provides for the grouping of pin directives into an entire vector. By "vector" we have meant and continue to mean "one set of values to apear on your chip's pins concurrently". For example, if your chip has 47 pins, one "test vector" designates 47 values. This interpetation of the term "test vector" implies a one-dimensional vector, NOT to be confused with a sequence of vectors that comprise an entire test. For us, an entire test running over many clock cycles is in fact a "sequence of vectors".

Because we have provided only for the specification of one pin at a time, there is as yet no concept of a test vector, or clock cycle. You use the third directive, literally

NEXT ;

to designate the end of one vector and the beginning of another.

This directive provides the "frames" around pin specifications to groupg them into vectors. Please note that between two NEXTs, all the pin/value specifications may appear in any order; this order must be irrelevant to the actual testing procedure.

<u>What About Clocks?</u>

This interchange format provides absolutely no distinction between clocks and other signals. A clock is merely a signal on some pin.

You introduce the distinction that clocks imply, namely that of separating time into discrete quanta, by using this "NEXT;" directive. That is, we expect that you will include in each test vector a specification for your clocks, both to turn the clock on and another to turn the clock off.

An entire clock cycle in a two-phase system will take four test vectors, (NEXTs):

one to turn ph1 ON

one to turn ph1 OFF

one to turn ph2 ON

one to turn ph2 OFF

That's A Lot of Test Vectors!

Four vectors per clock cycle seems steep alright, but each test vector may be very short in specification.

Because this format demands that you name pins explicitly in each test vector, as opposed to using a "positional" pin association, you may in fact omit the specification of some pins in many of the test vectors.

You need specify only those pins whose values you want to change. Pins which are meant to continue with their values from the previous test vector need not be respecified.

As some cultures say, the "drive pin" directive acts as a "sticky" switch.

Please ıemember that NEXT does not imply clocking of any sort, and in fact, the literal sequence

```
NEXT; NEXT;
```

is entirely equivalent to the shorter

```
NEXT;
```

For NEXTs to act like clocks, you must include between NEXTs at least one "drive pin" directive upon one of your clock signals.

Macro Calls

You may invoke a macro simply by writing the macro's name:

```
macro-name
```

This is entirely equivalent to writing instead the macro's body, the text enclosed between curly brackets ({}) in the macro's definition.

Initial Conditions

Upon starting up your test, you may assume that all your pins have been driven to 0.

Example

Imagine the world's second most simple chip, a single-bit of a counter.

It has inputs named RESET_IN and CARRY_IN, and outputs named RESET_OUT, CARRY_OUT, and VALUE.

Let's define the expected behavior in terms of synchronous logic:

```
RESET_OUT  =  RESET_IN           (pretty trivial eh?)

CARRY_OUT  =  VALUE  &  CARRY_IN

VALUE  =next   not( CARRY_IN )  &  VALUE   !
               CARRY_in  &  not( VALUE )
```

All this says is that

RESET_OUT follows RESET_IN all the time, and that

CARRY_OUT is the logical AND of the present VALUE held by this chip and CARRY_IN,   and that

this chip's VALUE, to be set upon the next clock cycle, remains unchanged if CARRY_IN is OFF, or flips if CARRY_IN is ON.

Now let's imagine the chip with its pads:

```
                      GND      CARRY_IN
             ----------+----------+-----------
             I                              I
             I                              I
     phl    -I                              I
             I                              I-  CARRY_OUT
 RESET_IN   -I                              I
             I                              I-  RESET_OUT
             I                              I
             I                              I-   VALUE
             ----------+----------+-----------
                      ph2         VDD
```

Here is a sample test specification, loaded with comments:

```
TOP:        0 , 11 ;       /* GND and CARRY_IN */

BOTTOM:   32,  1   ;       /* ph2 and VDD */

LEFT:     10,  31  ;       /* RESET_IN and Phl */

RIGHT:    21, 22, 20 ;     /* VALUE, RESET_OUT, CARRY_OUT*/

/* The pin numbers chosen are entirely arbitrary,

   except of course 0 (GND) and 1 (VDD).  We've chosen

  the clocks to be in the 30s, output in the 20s, and

  inputs in the teens.  */

MACRO:  PH1_ON { 31 = 1 }

MACRO:  PH1_OFF { 31 = 0 }

MACRO:  PH2_ON  { 32 = 1 }

MACRO:  PH2_OFF  { 32 = 0 }

MACRO:  RESET  { 10 = 1 }

MACRO:  UNRESET  { 10 = 0 }   /*(This is affecting

                                  RESET_IN*/

MACRO:  CARRY_IN  { 11 = 1 }

MACRO:  UN_CARRY_IN  { 11 = 0 }

/* This has been a randomly chosen set of macros.*/
```

```
BEGIN_TEST:
   UNRESET; UN_CARRY_IN: PH1_OFF; PH2_OFF:   NEXT:
   /* This first specification turns off a lot of
      signals.  The final NEXT; marks end-of-test-
      vector*/
   PH1_ON ;  NEXT ;   /* Diddle the clocks */
   PH1_OFF;  NEXT ;
   PH2_ON ;  NEXT ;
   PH2_OFF;  NEXT ;
/* Alright, let's get serious ... */
10 = 1;  PH1_ON;  NEXT;    /* Turn on RESET_IN*/
22 ? 1;  PH1_OFF; NEXT;    /* Check RESET_OUT */
/*Notice how we check RESET_OUT during a test vector
  strictly after that test-vector which set RESET_IN.
  The elements within a test-vector have no reliable
  "order of execution".  Thus, if we were to read
  RESET_OUT (22) during the same test-vector that
  set RESET_IN (10), we would not know what to
  expect.
  It is important to remember that NEXT has nothing
  to do with clock cycles.  The fact that we must
  read strictly after writing is a property of this
  testing scheme; it does not imply that your chip
  in fact imposes a delay between RESET_IN and
  RESET_OUT.  */
/* Keep the clocks moving ... */
```

```
      PH2_ON;   NEXT;

      PH2_OFF;  NEXT;

      /* Let's try setting CARRY_IN and see what that does

         to VALUE */

      11 = 1;  PH1_ON:  NEXT;

      21 ? 0;  PH1_OFF; NEXT;   /*Hope VALUE (21) has not

                                  changed.  We have not

                                  finished a complete

                                  clock-cycle*/

      PH2_ON;  NEXT;

      PH2_OFF; NEXT;   /* We now have completed a clock

                          cycle, and hence expect VALUE

                          to change */

      21 ? 1;

        11 = 0;  PH1_ON;  NEXT;

              /*We just checked VALUE, and simultaneously

                turned off CARRY_IN (11).  */

        /* We don't know which pin is affected first;

           CARRY_IN or PH1/  However, we can assume the

           CARRY-IN and PH1 have their specified values

           in place now, just after the NEXT. */

    END_TEST

End of Example
--------
$
```

Appendix "F"

Name

esim - event driven switch level simulator

Synopsis

```
esim [file1 [file2...]]
```

Description

Esim is an event-driven switch level simulator for NMOS translator circuits. *Esim* accepts commands from the user, executing each command before reading the next. Commands come in two flavors: those which manipulate the electrical network, and those to direct the simulation. Commands have the following simple syntax:

```
c arg1 arg2 ... argn <new line>
```

where 'c' is a single letter specifying the command to be performed and the *arg1* are arguments to that command. The arguments are separated by spaces (or tabs) and the command is terminated by a <new line>.

To run *esim* type,

```
esim file1 file2 ...
```

Esim will read and execute commands, first from *file1*, then *file2*, etc. If one of the file names is preceded by a '-', then that file becomes the new output file (the default output is stdout). For example,

```
esim f.sim -f.out g.sim
```

This would cause esim to read commands from f.sim, sending output to the default output. When f.sim was exhausted, f.out would become the new output file, and the commands in g.sim executed.

After all the files have been processed, and if the "q" command has not terminated the simulation run, esim will accept further commands from the user, prompting for each one like so:

```
sim>
```

The user can type individual commands or direct esim to another file using the "@" command:

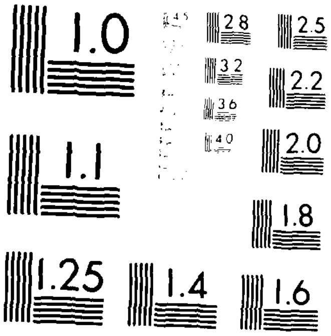
```
sim> @ patchfile.sim
```

This command would cause esim to read commands from "patchfile.sim", returning to interactive input when the file was exhausted.

It is common to have an initial network file prepared by a node extractor with perhaps a patch file or two prepared by hand. After reading these files into the simulator, the user would then interactively direct esim. This could be accomplished as follows:

```
esim file.sim patch.1 patch.2
```

AD-A151 961 SYSTEM DESIGN OF AUTOMATED VLSI (VERY LARGE SCALE 3/4
INTEGRATED) TEST STATIO..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. S TARIQ
UNCLASSIFIED DEC 84 AFIT/GE/EE/84D-27 F/G 9/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

After reading the files, esim would prompt for the first command. Or we could have typed:

```
% esim file.sim
sim> @  patch.1
sim> @  patch.2
```

Network Manipulation Commands

The electrical network to be simulated is made up of enhancement and depletion mode transistors interconnected by nodes. Components can be added to the network with the following commands.

e gate source drain

e gate source drain length width key xpos ypos area

Adds enhancement mode transistor to network with the specified gate, source, and drain nodes. The longer form includes size and location information as provided by the node extractor -- when making patches the short form is usually used.

d gate source drain

d gate source drain length width key xpos ypos area

Like "e" except for depletion mode devices.

C node1 node2 cap

Increase the capictance between node1 and node2 by cap. Esim ignores this unless either node1 or node2 is GND.

= node name1 name2 name3

Allows the user to specify synonyms for a given node. Used by the node extractor to relate user-provided node names to the node's internal name (usually just a number).

| comment ...

Lines beginning with vertical bar are treated as comments and ignored -- useful for deleting pieces of network in node extractor output files.

i node

Input record -- output by node extractor and not used by esim.

Currently, there is no way to remove components from the network once they have been added. You must go back to the input files and modify them (using the comment character) to exclude those components you wished removed. "N" records need not be included for new nodes the user wishes to patch into the network.

Simulator Commands

The user can specify which nodes are to have their values displayed after each simulation step:

w node1 -node2 node3 ...

Watch node1 and node3, stop watching node2. At the end of a simulation step, each watched node will be displayed like so:

```
node1=0 node3=X ...
```

To remove a node from the watched list, preface its name with a '-' in a "w" command.

W label node1 node2 ... noden

Watch bit vector. The values of nodes node1, ..., noden be will displayed as a bit vector:

```
label=010100   20
```

where the first 0 is the value of node1, the first 1 the value of node2, etc. The number displayed to right is the value of the bit vector interpreted as a binary number; this is omitted if the vector contains an X value. There is no way to unwatch a bit vector.

Before each simulation step the user can force nodes to be either high (1) or low (0) inputs (an input's value cannot be changed by the simulator!):

h node1 node2 ...

Force each node on the argument list to be a high input. Overrides previous input commands if necessary.

l node1 node2 ...

Like "h" except forces nodes to be a low input.

x node1 node2 ...

Removes nodes from whatever input list they happen to be on. The next simulation step will determine their correct value in the circuit. This is the default state of most nodes. Note that this does not force nodes to have an "X" value -- it simply removes them from the input lists.

The current value of a node can be determined in several ways:

v

View. Prints the values of all watched nodes and nodes on the high and low input lists.

? node1 node2 ...

Prints a synopsis of the named nodes including their current values and the state of all transistors that affect the value of these nodes. This is the most common way of wondering through the network in search of what went wrong ...

! node1 node2 ...

For each node in the argument list, prints a list of transistors controlled by that node.

"?" and "!" allow the user to go both backwards and forwards through the network in search of that piece causing all the problems.

The simulator is invoked with the following commands:

s

Simulation step. Propogates new values for the inputs through the network, returns when the network has settled. If things don't settle, command will never terminate -- try the "w" and "D" commands to narrow down the problem.

c

Cycle once through the clock, as defined by the K command.

I

Initialize. Circuits with state are often hard to initialize because the initial value of each node is X. To cure node whose value is charged-X and changes it to charged-0, then runs a simulation step. If one iterates the I command a couple times, this often leads to a stable initialized condition (indicated when an I command takes 0 events, i.e., the circuit is stable).
Try it -- if circuit does not become stable in 3 or 4 tries, this command is probably of no use.

Miscellaneous Commands

D

Toggle debug switch. Useful for debugging simulator and/or circuit. If debug switch is on, then during simulation step each time a watched node is encounted in some event, that fact is indicated to the user along with some event info. If a node keeps appearing in this printout, chances are that its value is oscillating. Vice versa, if your circuit never settles (i.e., it oscillates), you can use the "D" and "w" commands to find the node(s) that are causing the problem.

> filename

Write current state of each node into specified file. Useful for make a break point in your simulation run. Only stores values so isn't really useful to "dump" a run for later use -- see "<" command.

< filename

Read from specified file, reinitializing the value of each node as directed. Note that network must already exist and be identical to the network used to create the dump file with the ">" command. These state saving commands are really provided so that complicated initializing sequences need only be simulated once.

L

Invokes network processor that finds all subnets corresponding to simple logic gates and converts them into form that allows faster simulation. Often it does the right thing, leading to a 25% to 50% reduction in the time for a single step. [We know of one case where the transformation was not transparent, so caveat simulee...]

X ...

call estension command -- provides for user extensions to simulator.

q

exit to system.

Local Extensions

V node vector

Define a vecto: of inputs for the node. The first element is initially set as the input for node. Set the next element of the vector as the input after a cycle.

R n

Run the simulator through n cycles. If n is not present make the run as long as the longest vector. All watch nodes are reported back as vectors.

N

Clear all previously defined input vectors.

K node1 vector1 node2 vector2 ... nodeN vectorN

Define the clock. Each cycle, nodes 1 through N must run through their respective vectors.

Author

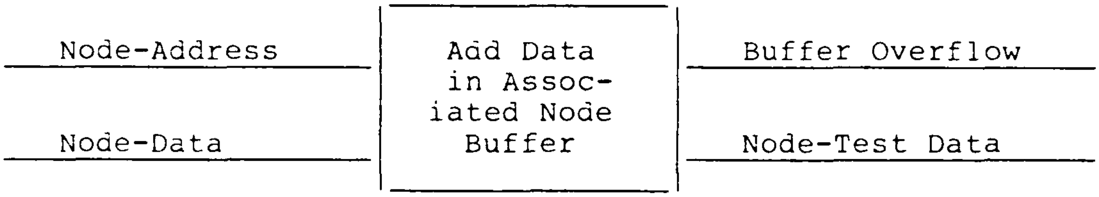
Chris Terman

PROCESS DEFINITION

PROCESS NAME: Add Data in Associated Node Buffer

PROCESS ID NUMBER: A1313

PROCESS PICTURE:



PROCESS DESCRIPTION: This process gauges the incoming data and its associated node buffer to confirm availability of space. This process generates "buffer overflow" if incoming data exceeds the available space. Otherwise data is added to data already held in the buffer.

INPUT DATA FLOW: Node-Address
Node-Data

OUTPUT DATA FLOW: Buffer Overflow
Node-Test Data

REFERENCE DIAGRAM: A131

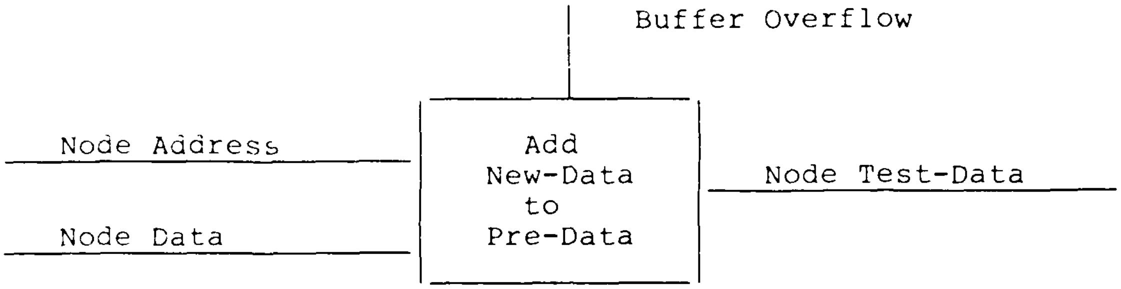
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Add New-Data to Pre-Data

PROCESS ID NUMBER: A13134

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, if total data (new-data and pre-data) does not exceed the buffer capacity, adds new-data to the Pre-data in memory buffer.

INPUT DATA FLOW: Node Address
Node Data

OUTPUT DATA FLOW: Node Test Data

REFERENCE DIAGRAM: A1313

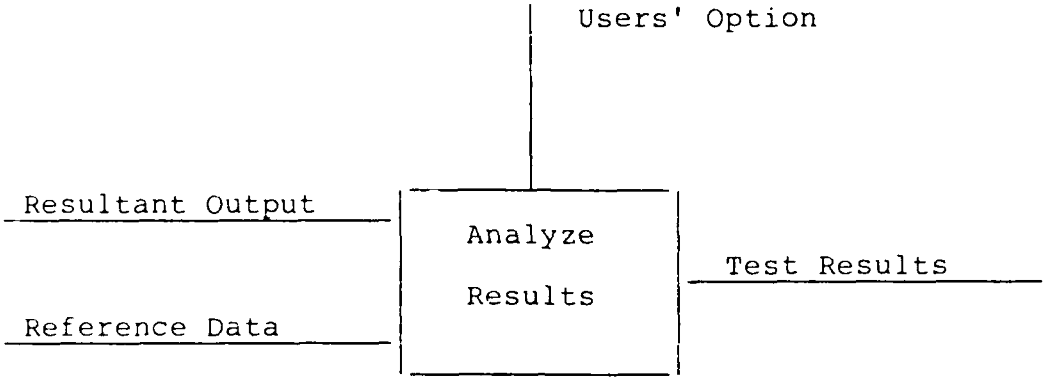
ADDITIONAL COMMENTS: This process is readily implemented in "C" language by a system library routine (strcat (S_1, S_2)), which cancetenates two given buffers.

PROCESS DEFINITION

PROCESS NAME: Analyze Results

PROCESS ID NUMBER: A41

PROCESS PICTURE:



PROCESS DESCRIPTION: This process transforms the resultant output from IC tester pins domain to ICUT pins' domain and compares the output with reference data for any non-conformity. It generates an additional message on successful/unsuccessful completion of test.

INPUT DATA FLOW: Resultant Output

Reference Data

OUTPUT DATA FLOW: Test Results

REFERENCE DIAGRAM: A4

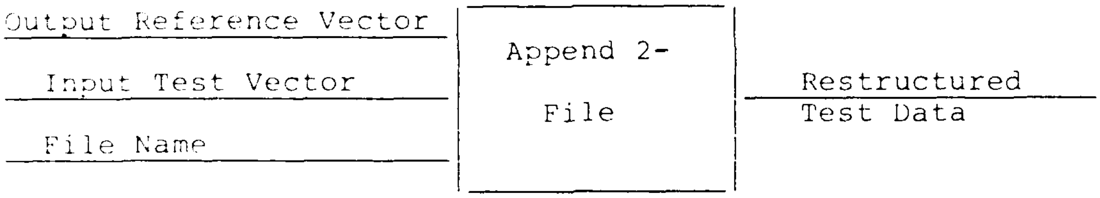
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Append2 - File

PROCESS ID NUMBER: A133

PROCESS PICTURE:



PROCESS DESCRIPTION: This process empties the buffers to an external file "filename", and is implemented by a system library routine. (fprintf)

INPUT DATA FLOW: Output Reference Vector

Input Test Vector

File Name

OUTPUT DATA FLOW: Restructured Test Data

REFERENCE DIAGRAM: A13

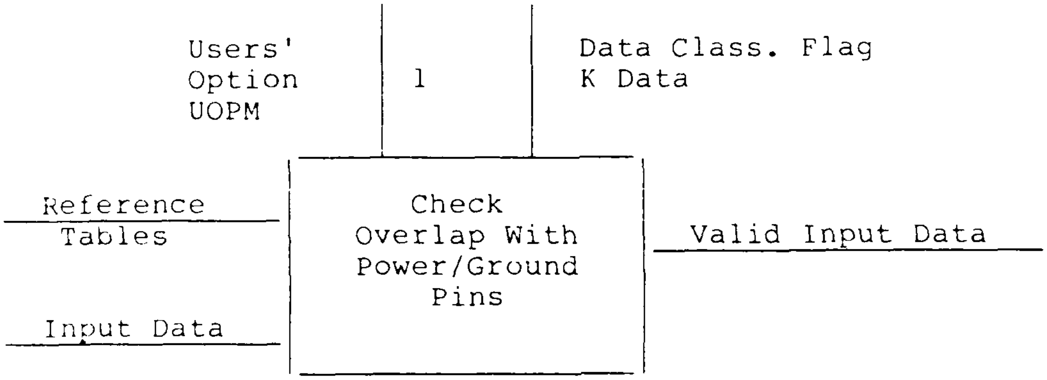
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check Overlap with Power/Ground Pins

PROCESS ID NUMBER: A252

PROCESS PICTURE:



PROCESS DESCRIPTION: This process carries out bitwise comparison of input data (initialization and test data in Manual Mode of operation) to confirm that no pin designated as "Pcwer/Ground" pin is simulated by the input data.

INPUT DATA FLOW: Reference Tables

Input Data

OUTPUT DATA FLOW: Valid Input Data

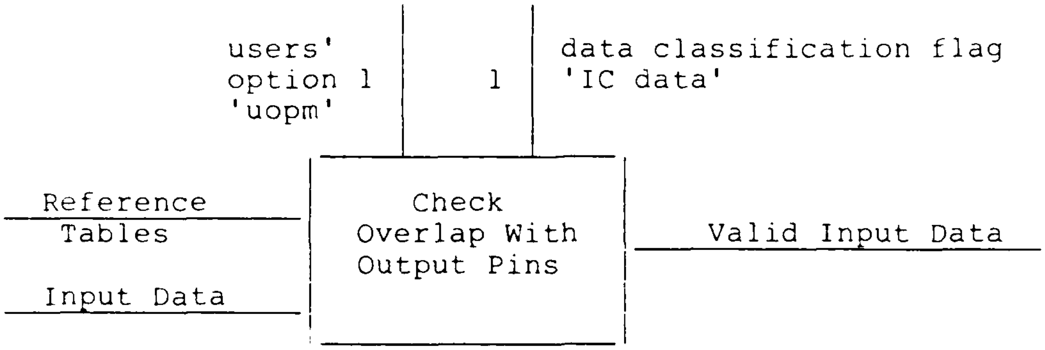
REFERENCE DIAGRAM: A25

PROCESS DEFINITION

PROCESS NAME: Check Overlap with Output Pins

PROCESS ID NUMBER: A251

PROCESS PICTURE:



PROCESS DESCRIPTION: This process carries out bitwise comparison of init data vector in Manual mode of operation to confirm that no designated "output" pin is being simulated by the init data vector.

INPUT DATA FLOW: Reference Tables

Input Data (init/test data)

*init = initialization

OUTPUT DATA FLOW: Valid Input Data

REFERENCE DIAGRAM: A25

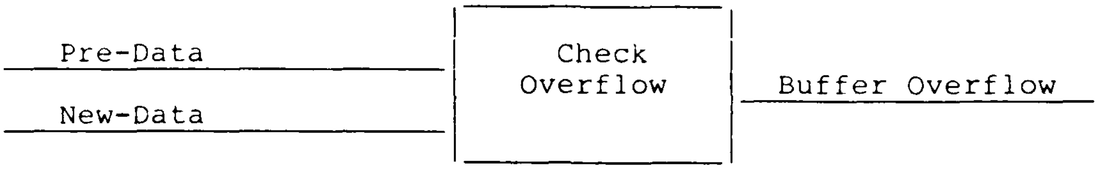
ADDITIONAL COMMENTS: This process is activated only during "Manual" mode of operation.

PROCESS DEFINITION

PROCESS NAME: Check Overflow

PROCESS ID NUMBER: A13133

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, generates an error signal "buffer overflow" if the total amount of data, pre-data and new-data exceeds the buffer capacity.

INPUT DATA FLOW: Pre-Data

New-Data

OUTPUT DATA FLOW: Buffer-Overflow

REFERENCE DIAGRAM: A1313

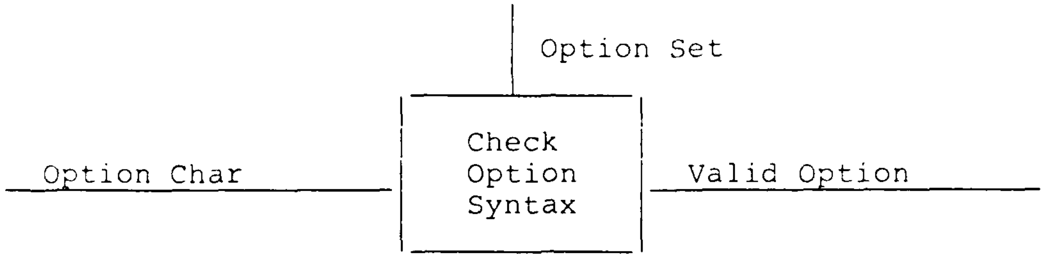
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check Option Syntax

PROCESS ID NUMBER: A2233

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, verifies that option-input-character is within "A to D" for manual mode and within range of "A to F" for Auto mode of operation. In case of non-validity, it asks the user to input a valid response.

INPUT DATA FLOW: Option Char

OUTPUT DATA FLOW: Valid Option

REFERENCE DIAGRAM: A223

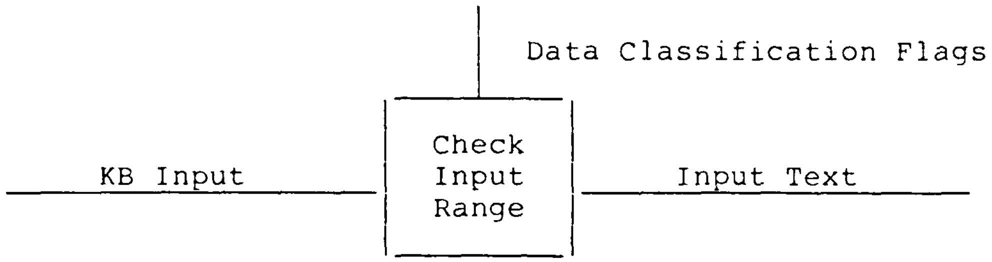
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check Input Range

PROCESS ID NUMBER: A222

PROCESS PICTURE:



PROCESS DESCRIPTION: This process checks that input received in response to a particular system response lies within the expected range. The range is made known to the user in user-friendly system menues.

INPUT DATA FLOW: KB Input

OUTPUT DATA FLOW: Input Text

REFERENCE DIAGRAM: A22

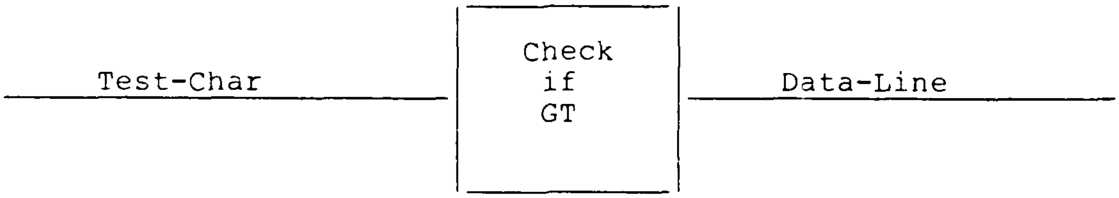
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check if GT

PROCESS ID NUMBER: A1123

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, on receiving a test-char (first character of a given text-line) checks if it is ">" (greater than sign). If it is found to be true then text-liine is declared to be a data-line.

INPUT DATA FLOW: Test-Char

OUTPUT DATA FLOW: Data-Line

REFERENCE DIAGRAM: A112

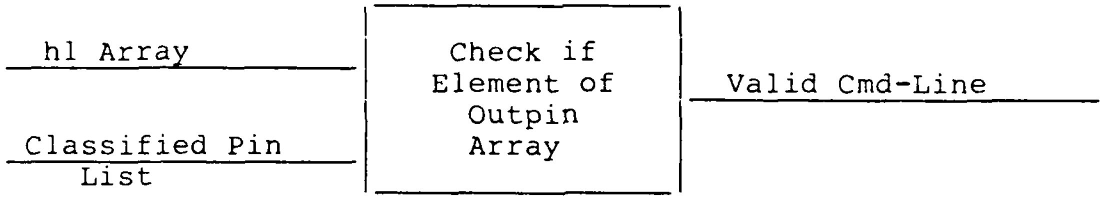
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check if Element of Outpin-Array

PROCESS ID NUMBER: A12312

PROCESS PICTURE:



PROCESS DESCRIPTION: This function compares each element of hl array successively with all elements of outpin array. It sets a flag for cmd-line being valid if no element is found to be common between two arrays.

INPUT DATA FLOW: hl Array

Classified Pin List

OUTPUT DATA FLOW: Valid Cmd-Line

REFEKENCE DIAGRAM: A1231

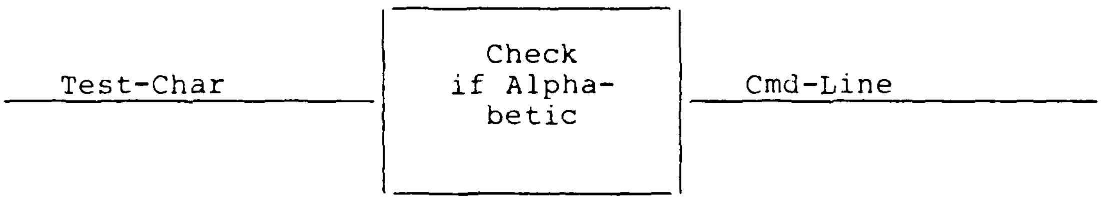
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check if Alphabetic

PROCESS ID NUMBER: A1122

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, on receiving a test-char checks if it is alphabetic (a-z or A-Z) by a system library routine (isalpha()). If test-char is found to be alphabetic, text line is declared to be a cmd-line (command-line).

INPUT DATA FLOW: Test-Char

OUTPUT DATA FLOW: Cmd-Line

REFERENCE DIAGRAM: A112

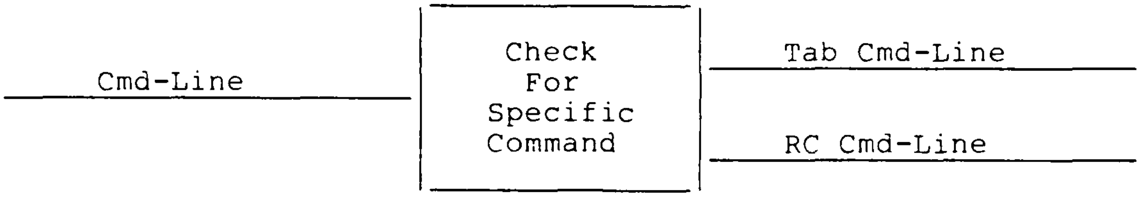
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check for Specific Command

PROCESS ID NUMBER: A121

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, interprets the first character of a cmd-line and categorizes it to be tab cmd-line, if it is w, V, k, or I. Otherwise, for first character to be h. l, or N cmd-line is categorized to be RC Cmd-Line.

INPUT DATA FLOW: Cmd-Line

OUTPUT DATA FLOW: Tab Cmd-Line
RC Cmd-Line

REFERENCE DIAGRAM: A121, A12

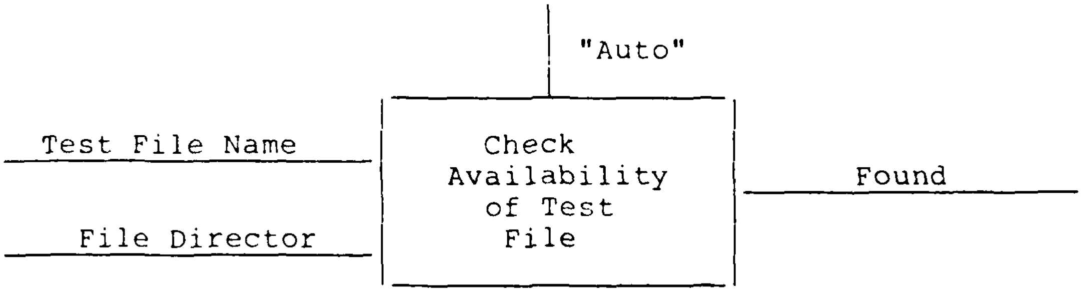
ADDITIONAL COMMENTS: The details of ESIM commands are described in Appendix "F".

PROCESS DEFINITION

PROCESS NAME: Check Availability of Test File

PROCESS ID NUMBER: A2221

PROCESS PICTURE:



PROCESS DESCRIPTION: This process in Auto mode operation scans file directory to check availability of test file whose name has been inputted by the user. Flag "found" is set to be true if file is available.

INPUT DATA FLOW: Test File Name

OUTPUT DATA FLOW: Found

REFERENCE DIAGRAM: A222

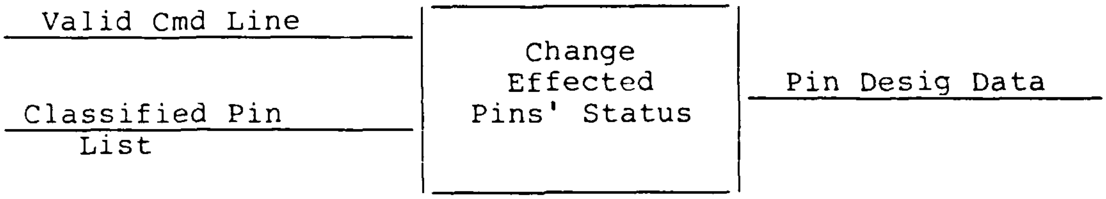
ADDITIONAL COMMENTS: This process is readily implemented in "C" language by system library function (open (...)), which accesses a given file in lead, write or append mode.

PROCESS DEFINITION

PROCESS NAME: Change Effected Pins' Status

PROCESS ID NUMBER: A1232

PROCESS PICTURE:



PROCESS DESCRIPTION: This process changes the status of a pin to "high/low". For a valid Cmd.line. The effected pins must belong to either class of input pins or class of unmarked pins.

INPUT DATA FLOW: Valid Cmd.Line
Classified Pin Lists

OUTPUT DATA FLOW: Pin Desig Data

REFERENCE DIAGRAM: A 123

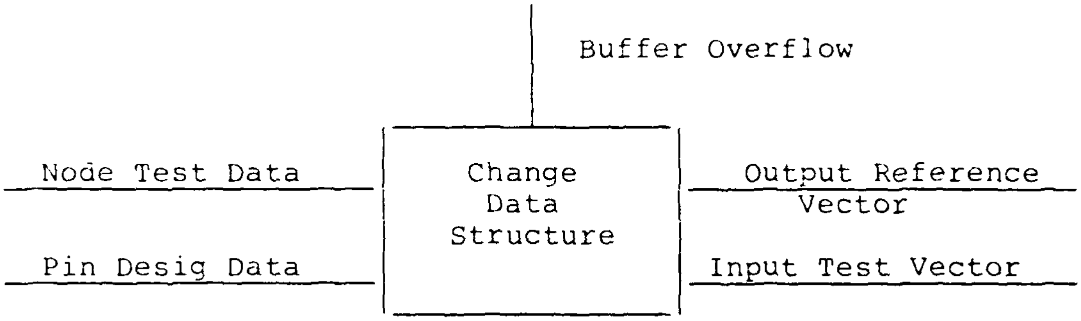
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Change Data Structure

PROCESS ID NUMBER: A132

PROCESS PICTURE:



PROCESS DESCRIPTION: This process converts the node test data for all input pins to input test vector and for all output pins to output reference vector. This process is activated by buffer overflow to empty the buffers for new data.

INPUT DATA FLOW: Node Test Data
Pin Desig Data

OUTPUT DATA FLOW: Output Reference Vector
Input Test Vector

REFERENCE DIAGRAM: A13

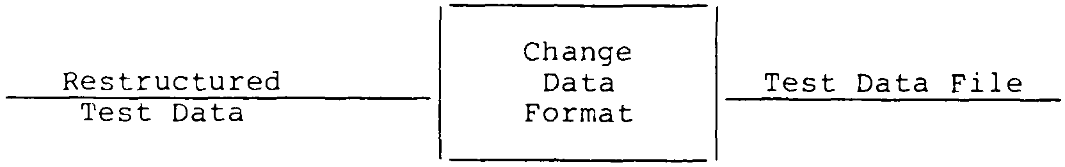
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Change Data Format

PROCESS ID NUMBER: A14

PROCESS PICTURE:



PROCESS DESCRIPTION: This process changes the memory storage pattern of restructured test data from VAX-system format into LSI-11 micro computer data-format, and transfers this file onto an 8" floppy disk. This is implemented through system library routines.

INPUT DATA FLOW: Restructured Test Data

OUTPUT DATA FLOW: Test Data File

REFERENCE DIAGRAM: A1, A14

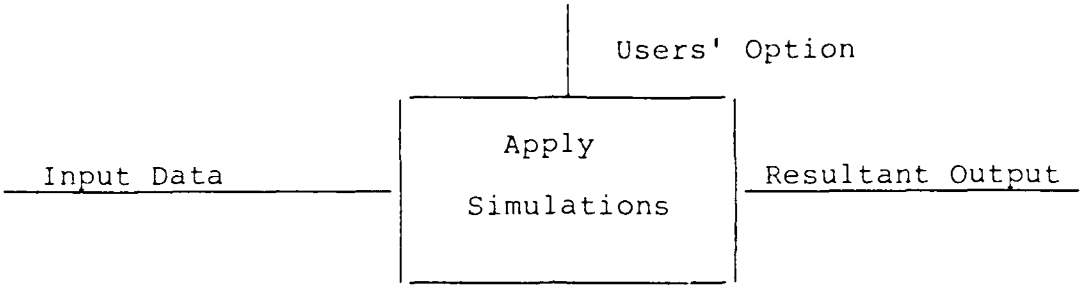
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Apply Simulations

PROCESS ID NUMBER: A3

PROCESS PICTURE:



PROCESS DESCRIPTION: This process translates input data from IC pin numbers to their respective physical locations on IC tester. It converts the simulation data into "SIEVE" format which is particularly required to operate Stanford IC tester. This process also applies physical voltages to effect simulation of IC Under Test (ICUT) and sample the pins of ICUT to get "resultant output".

INPUT DATA FLOW: Input data - input data consists of Manual test data entered through keyboard in "Manual" mode of operation or test data received from test data file in "Auto" mode of operation.

OUTPUT DATA FLOW: Resultant output.

REFERENCE DIAGRAM: A0

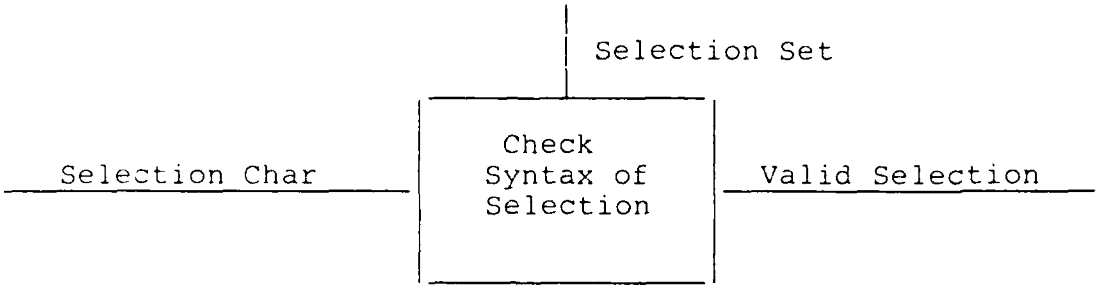
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Check Syntax of Selection

PROCESS ID NUMBER: A2313

PROCESS PICTURE:



PROCESS DESCRIPTION: This process confirms that selection char, input by user, actually has within range of offered selections.

INPUT DATA FLOW: Selection Char

OUTPUT DATA FLOW: Valid Selection

REFERENCE DIAGRAM: A231

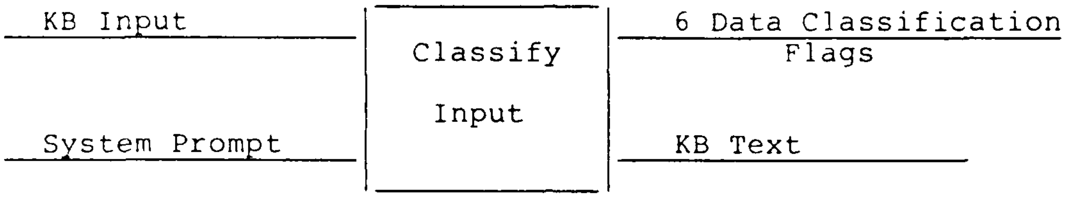
ADDITIONAL COMMENTS: This process repeats itself to get valid input from user.

PROCESS DEFINITION

PROCESS NAME: Classify Input

PROCESS ID NUMBER: A22

PROCESS PICTURE:



PROCESS DESCRIPTION: This process classifies all keyboard input into three broad categories of option data, IC data and test data. It also sets six data classification flags in accordance with expected response to a particular system prompt.

INPUT DATA FLOW: KB Input

System Prompt (generated by executive software program)

OUTPUT DATA FLOW: Six Data Classification Flags

KB test

REFERENCE DIAGRAM: A2

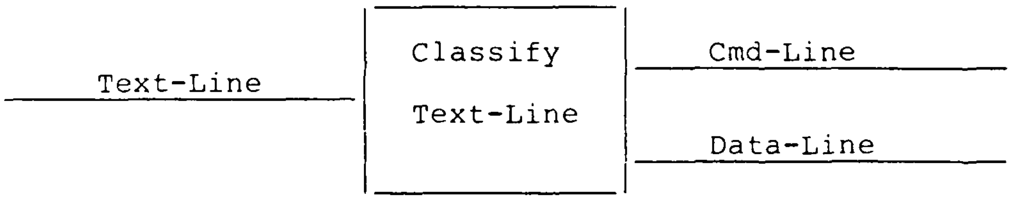
ADDITIONAL COMMENTS: Details of data classification flags is included in description of node A22 & A221.

PROCESS DEFINITION

PROCESS NAME: Classify Text-Line

PROCESS ID NUMBER: A112

PROCESS PICTURE:



PROCESS DESCRIPTION: This process categorizes each text-line lead from ESIm file to be either a Cmd-line (first character being an alphabet) or a data-line (first character being a '>'). It ignores the blank lines.

INPUT DATA FLOW: Text-Line

OUTPUT DATA FLOW: Cmd-Line
Data-Line

REFERENCE DIAGRAM: A11

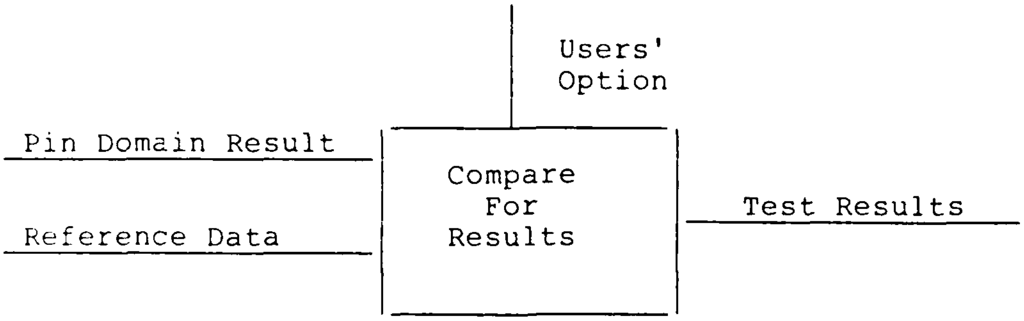
ADDITIONAL COMMENTS: A brief explanation of commands and data format of ESIM file is attached as Appendix "F".

PROCESS DEFINITION

PROCESS NAME: Compare for Results

PROCESS ID NUMBER: A412

PROCESS PICTURE:



PROCESS DESCRIPTION: This process carries out bitwise comparison between pin domain results (output of ICUT) and reference data to single out any non-conformity. It generates a "GO/NOGO" message for successful/ unsuccessful completion of a test.

INPUT DATA FLOW: Pin Domain Result

Reference Data

OUTPUT DATA FLOW: Test Results

REFERENCE DIAGRAM: A41

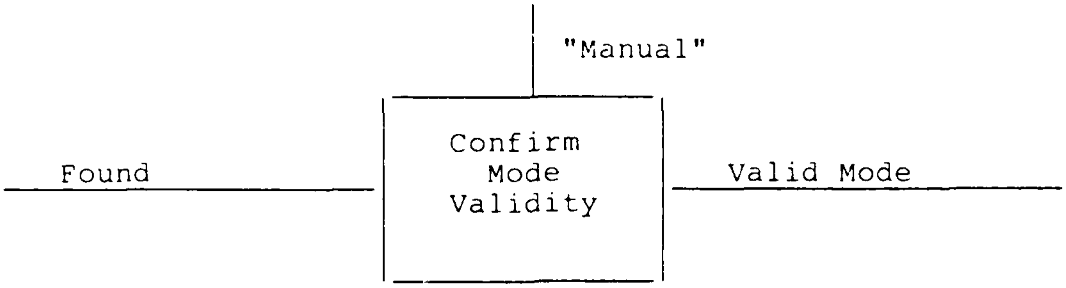
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Confirm Mode Validity

PROCESS ID NUMBER: A2222

PROCESS PICTURE:



PROCESS DESCRIPTION: This process confirms the validity of mode if found is true in Auto-mode or if "Manual" mode has been preferred.

INPUT DATA FLOW: Found

OUTPUT DATA FLOW: Valid Mode

DIAGRAM REFERENCE: A222

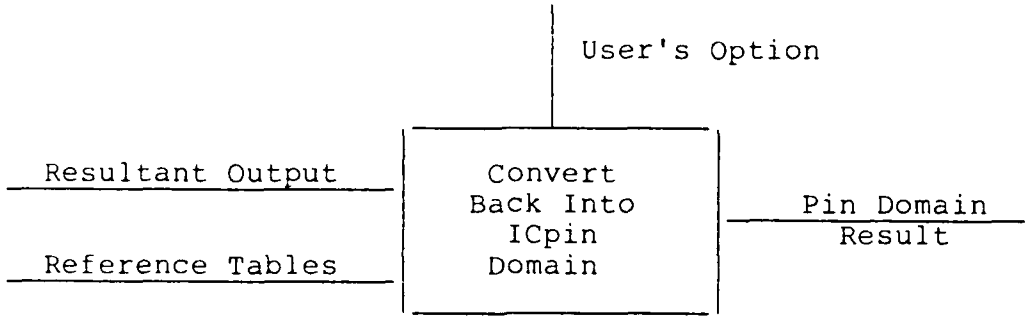
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Convert Back into ICpin Domain

PROCESS ID NUMBER: A411

PROCESS PICTURE:



PROCESS DESCRIPTION: This process converts the sampled output of ICUT into ICpin domain by referring reference tables which establish correspondence between tester pins and ICUT pins.

INPUT DATA FLOW: Resultant Output

Reference Tables

OUTPUT DATA FLOW: Pin Domain Result

REFERENCE DIAGRAM: A41

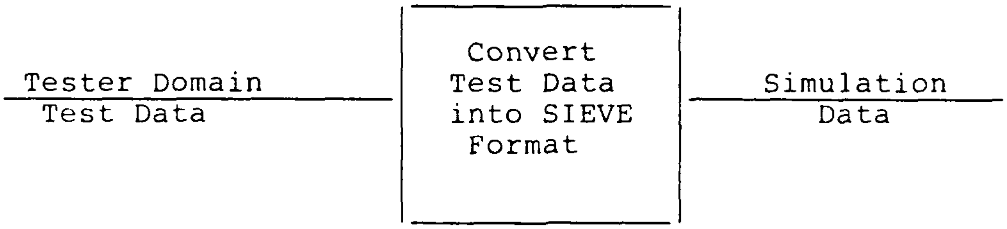
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAMF: Convert Test Data into SIEVE Format

PROCESS ID NUMBER: A314

PROCESS PICTURE:



PROCESS DESCRIPTION: This process changes the format of test data in "SIEVE" format. SIEVE data format is specifically required to simulate Stanford IC Tester.

INPUT DATA FLOW: Tester Domain Test Data

OUTPUT DATA FLOW: Simulation Data

REFERENCE DIAGRAM: A31

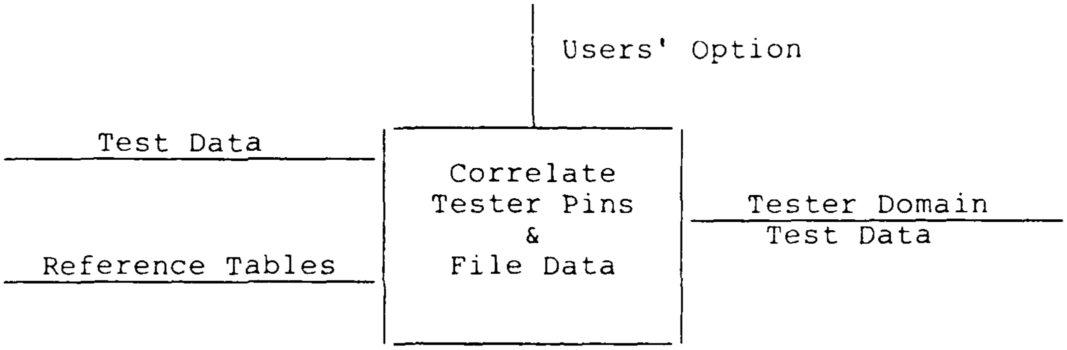
ADDITIONAL COMMENTS: An explanation of "SIEVE" data format is attached as Appendix "E".

PROCESS DEFINITION

PROCESS NAME: Correlate Tester Pins & File Data

PROCESS ID NUMBER: A313

PROCESS PICTURE:



PROCESS DESCRIPTION: This process establishes correspondence between test data received from test data file in "Auto" mode of operation and physical location of ICUT pins on IC tester.

INPUT DATA FLOW: Test Data
Reference Tables

OUTPUT DATA FLOW: Tester Domain Test Data

REFERENCE DIAGRAM: A31

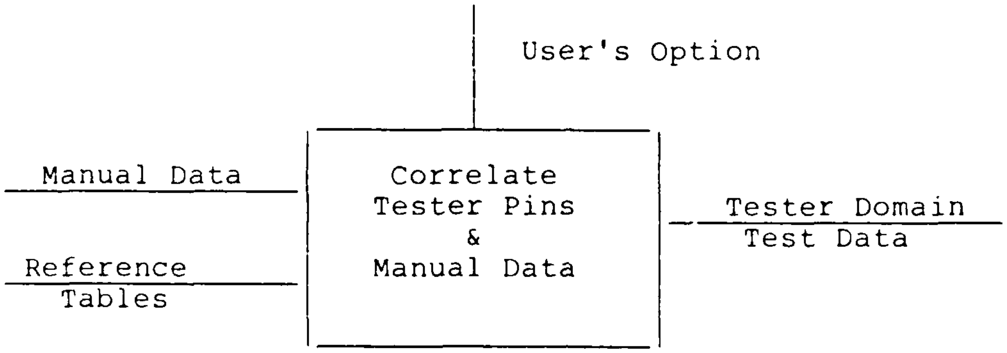
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Correlate Tester Pins & Manual Data

PROCESS ID NUMBER: A312

PROCESS PICTURE:



PROCESS DESCRIPTION: This process establishes correspondence between the test data received from keyboard in "Manual" mode of operation and physical location of ICUT pins on IC tester.

INPUT DATA FLOW: Manual Data

Reference Tables

OUTPUT DATA FLOW: Tester Domain Test Data

REFERENCE DIAGRAM: A31

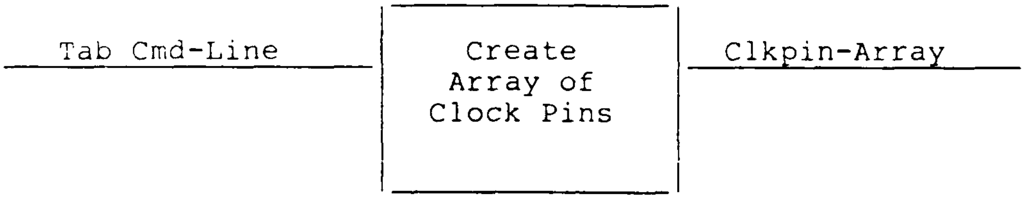
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Create Array of Clock Pins

PROCESS ID NUMBER: A1222

PROCESS PICTURE:



PROCESS DESCRIPTION: This process on interpreting first character of incoming command line "cmd-line" to be 'K', reads the remaining line to set up an array of clock pins and data associated with each pin.

INPUT DATA FLOW: Tab Cmd-line.

OUTPUT DATA FLOW: Clkpin-array (array containing names of clocking pins)

REFERENCE DIAGRAM: A122

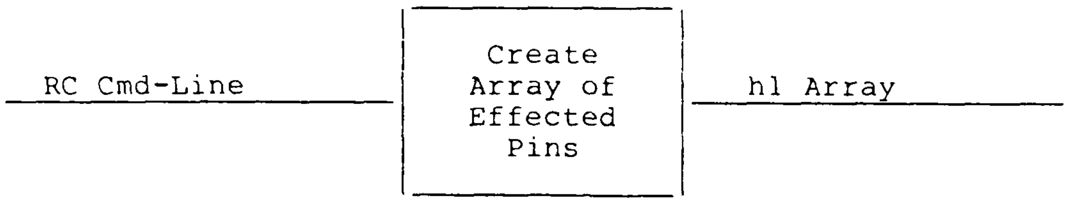
ADDITIONAL COMMENTS: A typical example of ESIM file showing various "cmd-lines" is included in Appendix "F" (Page F-6).

PROCESS DEFINITION

PROCESS NAME: Create Array of Effected Pins

PROCESS ID NUMBER: A12311

PROCESS PICTURE:



PROCESS DESCRIPTION: This function on receiving a RC Cmd-Line checks the first character to be "h" or "l". On conformation it generates an array of pin-names included in the remaining RC Cmd-Line.

INPUT DATA FLOW: RC Cmd-Line

OUTPUT DATA FLOW: hl Array

REFERENCE DIAGRAM: A1231

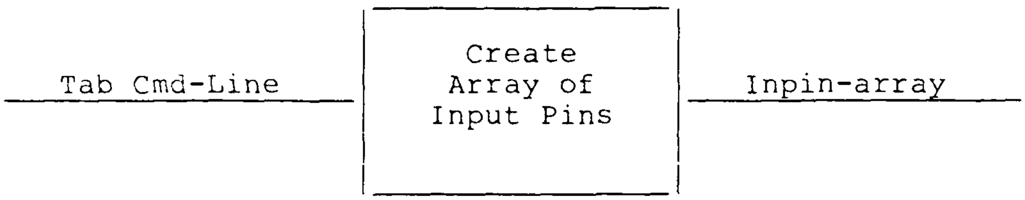
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Create Array of Input Pins

PROCESS ID NUMBER: A1223

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, on interpreting first character of incoming command line to be "V", sets up an array of input pins and adds the name and data included in the command line to the inpin array.

INPUT DATA FLOW: Tab Cmd-line

OUTPUT DATA FLOW: Inpin-array (array containing names of input pins)

REFERENCE DIAGRAM: A122

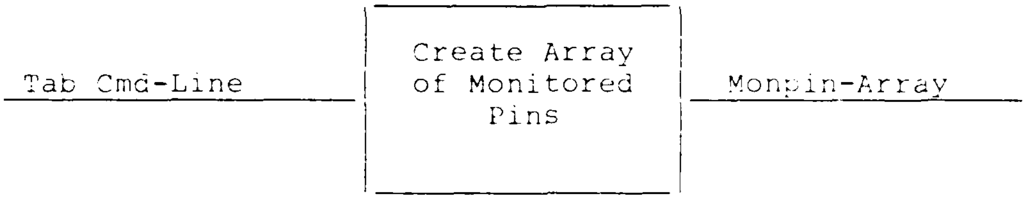
ADDITIONAL COMMENTS: A typical example of ESIM file showing various "cmd-lines" is included in Appendix "F" (Page F-6).

PROCESS DEFINITION

PROCESS NAME: Create Array of Monitored Pins

PROCESS ID NUMBER: A1221

PROCESS PICTURE:



PROCESS DESCRIPTION: This process on interpreting first character of incoming cmd-line to be "W" reads the pin names in the remaining command-line and sets up an array containing names of the monitored pins.

INPUT DATA FLOW: Tab Cmd-line

OUTPUT DATA FLOW: Monpin-array (array containing names of monitored pins)

REFERENCE DIAGRAM: A122

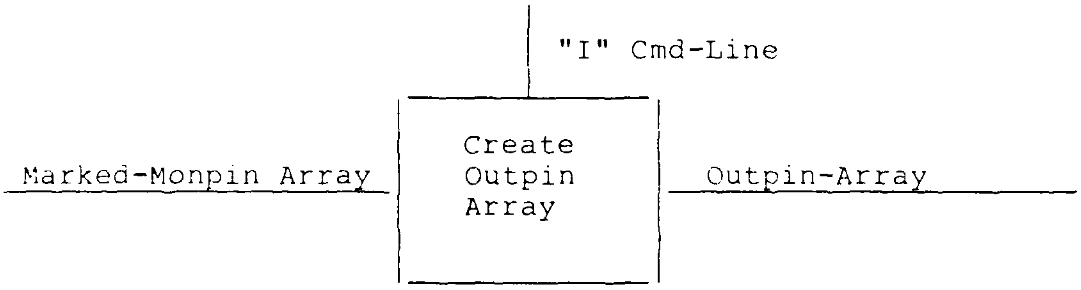
ADDITIONAL COMMENTS: A typical example of ESIM file showing various "Cmd-lines" is included in Appendix "F" (Page F-6).

PROCESS OUTPIN ARRAY

PROCESS NAME: Create Outpin Array

PROCESS ID NUMBER: A12243

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans the monpin-array with marked input/clock pins. It segregates the unmarked pins as output pins and generates "outpin-array" of output pins.

INPUT DATA FLOW: Marked-Monpin-Array

OUTPUT DATA FLOW: Outpin-Array

REFERENCE DIAGRAM: A1224

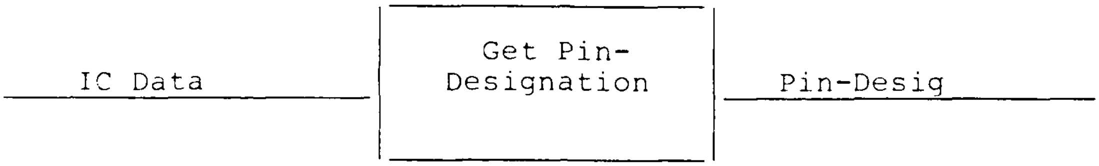
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Get Pin-Designations

PROCESS ID NUMBER: A2321

PROCESS PICTURE:



PROCESS DESCRIPTION: This process asks the user to designate all pin numbers successively in a selected pin-table (e.g., enter name of pinl = etc.) Valid response to this process are pin-name or an X (don't care) response.

INPUT DATA FLOW: IC Data

OUTPUT DATA FLOW: Pin-Desig

REFERENCE DIAGRAM: A232

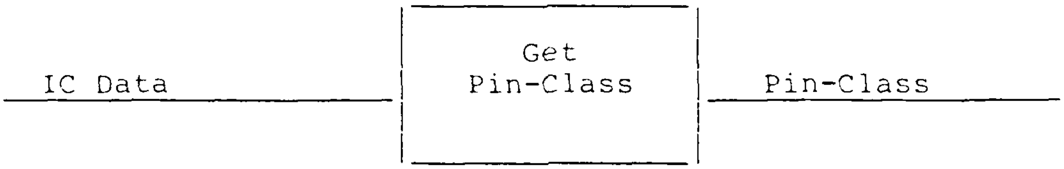
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Get Pin-Class

PROCESS ID NUMBER: A2322

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, asks the user to input class of each pin successively in a selected pin-table (e.g., enter class for pinl = etc.). Valid response to this prompt are X - don't care, I - input, O - output, K - clock, P - power, or G- ground.

INPUT DATA FLOW: IC Data

OUTPUT DATA FLOW: Pin-Class

REFERENCE DIAGRAM: A232

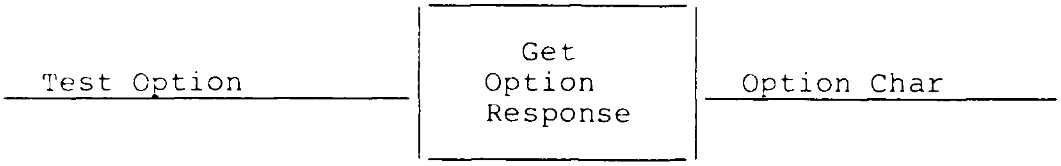
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Get Option Response

PROCESS ID NUMBER: A2232

PROCESS PICTURE:



PROCESS DESCRIPTIONI: This process, receives one of the selections (A to F for Auto mode and A to D for Manual mode as narrated in description of Node A22) from the keyboard.

INPUT DATA FLOW: Test Option

OUTPUT DATA FLOW: Option Char

REFERENCE DIAGRAM: A223

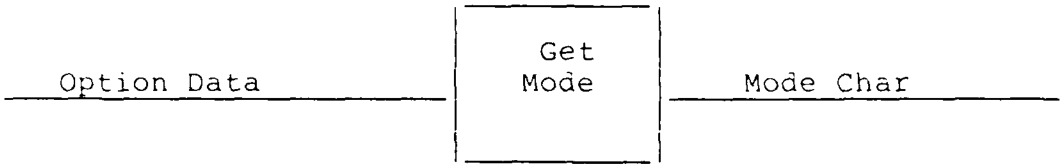
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Get Mode

PROCESS IDNUMBER: A2211

PROCESS PICTURE:



PROCESS DESCRIPTION: This process receives a character "A or M" for Auto or Manual mode selection respectively. This character is input by user then keyboard in response to system prompt.

INPUT DATA FLOW: Option Data

OUTPUT DATA FLOW: Mode Char

REFERENCE DIAGRAM: A221

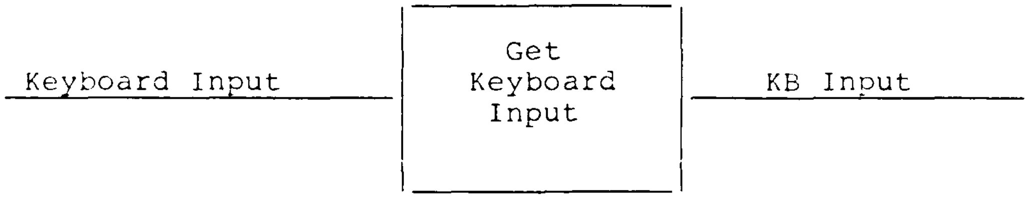
ADDITIONAL COMMENTS: This process is readily implemented in "C" language by system library function (get char()).

PROCESS DEFINITION

PROCESS NAME: Get Keyboard Input

PROCESS ID NUMBER: A21

PROCESS PICTURE:



PROCESS DESCRIPTION: This process gets any character or character string input then keyboard by the user as a command or data input in response to program prompts. This process is implemented by system library functions. (fget, fgets, & fscan)

INPUT DATA FLOW: Keyboard Input

OUTPUT DATA FLOW: KB Input

REFERENCE DIAGRAM: A2, A21

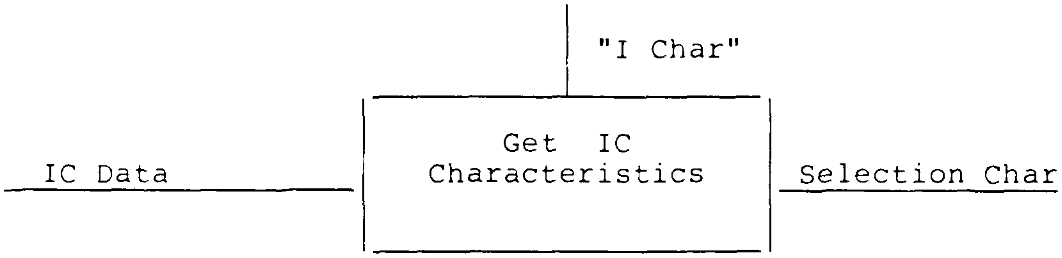
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Get IC Characteristics

PROCESS ID NUMBER: A2313

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, gets the selection character from keyboard which is keyed in by the user in response to program menu.

INPUT DATA FLOW: IC Data

OUTPUT DATA FLOW: Selection Char

REFERENCE DIAGRAM: A231

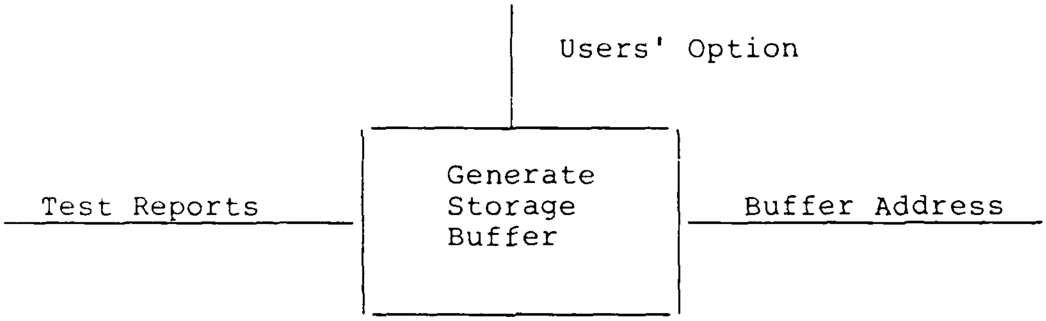
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Generate Storage Buffer

PROCESS ID NUMBER: A422

PROCESS PICTURE:



PROCESS DESCRIPTION: This process depending on users' option to store test reports in an external file, sets up a storage buffer and adds all test reports till completion of test.

INPUT DATA FLOW: Test Reports

OUTPUT DATA FLOW: Buffer Address

REFERENCE DIAGRAM: A42

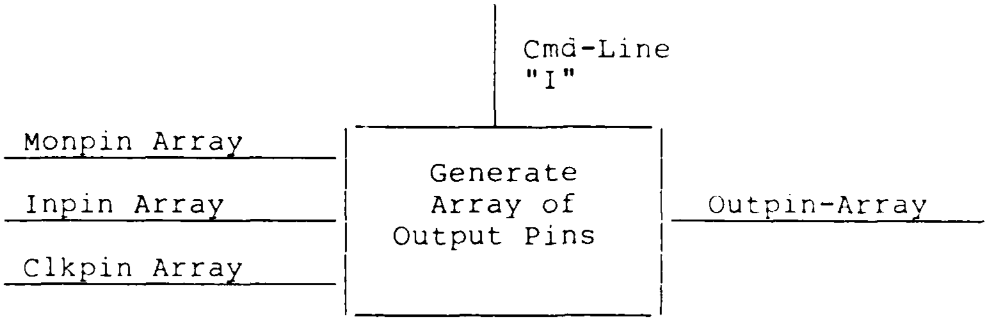
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Generate Array of Output Pins

PROCESS ID NUMBER: A1224

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, on receiving a control command, establishes a cross-reference between elements of inpin/clkpin arrays and monpin-array. It then, generates another array from elements of monitored pins, not marked as input/clock pins. The new array is named "outpin-array".

INPUT DATA FLOW: Monpin-Array

Inpin-Array

Clkpin-array

OUTPUT DATA FLOW: Outpin-Array

REFERENCE DIAGRAM: A122

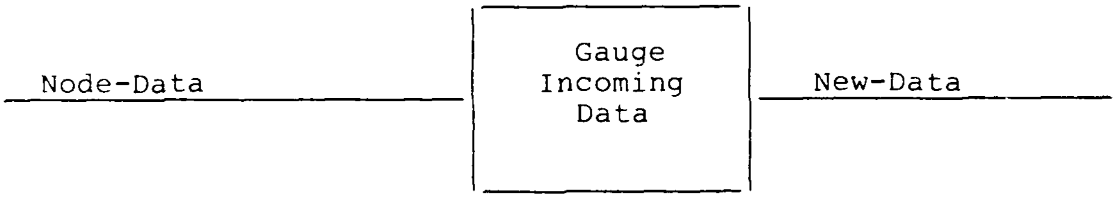
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Gauge Incoming Data

PROCESS ID NUMBER: A13132

PROCESS PICTURE:



PROCESS DESCRIPTION: This process measures the amount of data in bytes in a given data-line [A data-line consists of two parts, i.e., name (node-name) and data (node-data)].

INPUT DATA FLOW: Node-Data

OUTPUT DATA FLOW: New-Data

REFERENCE DIAGRAM: A1313

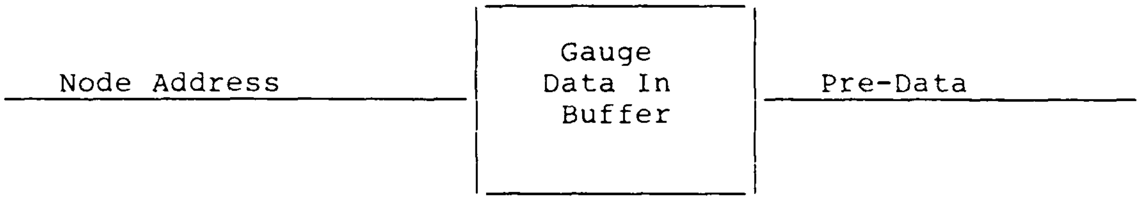
ADDITIONAL COMMENTS: This process is readily implemented in "C" language by a system library routine (strlen (S)) which returns an integer giving total amount of data in bytes in a given buffer.

PROCESS DEFINITION

PROCESS NAME: Gauge Data in Buffer

PROCESS ID NUMBER: A13131

PROCESS PICTURE:



PROCESS DESCRIPTION: This process measures the amount of data already available in a given node buffer whose address is passed-in.

INPUT DATA FLOW: Node Address

OUTPUT DATA FLOW: Pre-Data

REFERENCE DIAGRAM: A1313

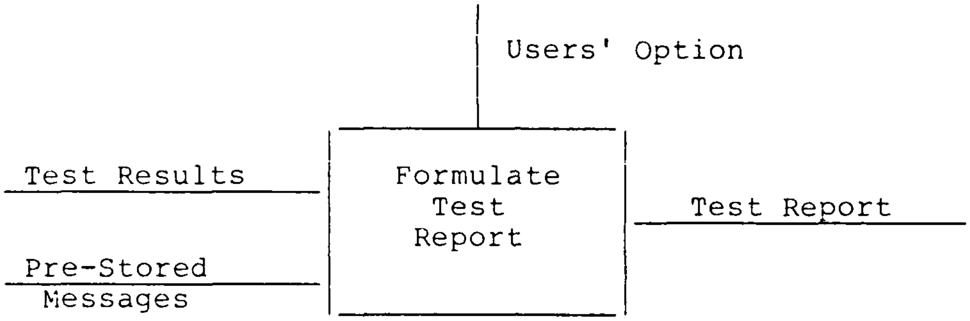
ADDITIONAL COMMENTS: This process is readily implemented in "C" language by a system library routine (strlen (S)) which returns an integer, giving total amount of data in bytes, held in a memory buffer.

PROCESS DEFINITION

PROCESS NAME: Formulate Test Report

PROCESS ID NUMBER: A421

PROCESS PICTURE:



PROCESS DESCRIPTION: This process generates test report from test results and pre-stored messages, depending on successful/unsuccessful completion of test.

INPUT DATA FLOW: Test Results

Pre-Stored Messages

OUTPUT DATA FLOW: Test Results

REFERENCE DIAGRAM: A42

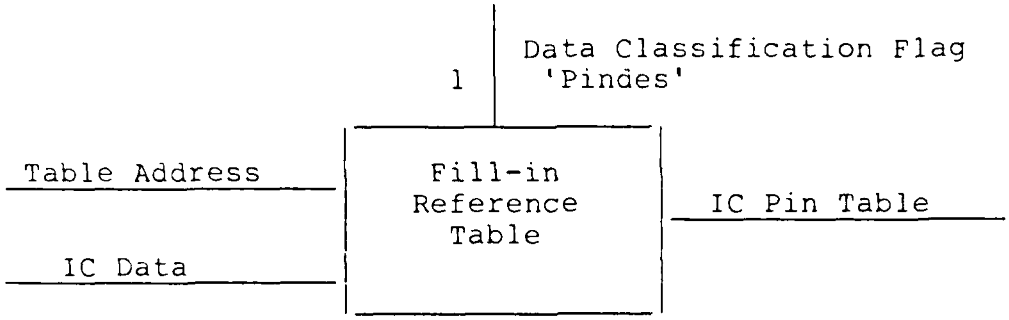
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Fill-in Reference Table

PROCESS ID NUMBER: A242

PROCESS PICTURE:



PROCESS DESCRIPTION: This process receives information regarding pin numbers, pin designations and pin class for an ICUT and finn-0in the selected pre-stored table.

INPUT DATA FLOW: Table Address - 'Address of selected pre-stored table"

IC Data - Information like

Pin 14 = Vcc P

Pin 7 = Gnd G

Pin 4 = K I etc.

OUTPUT DATA FLOW: IC Pin Table

REFERENCE DIAGRAM: A24

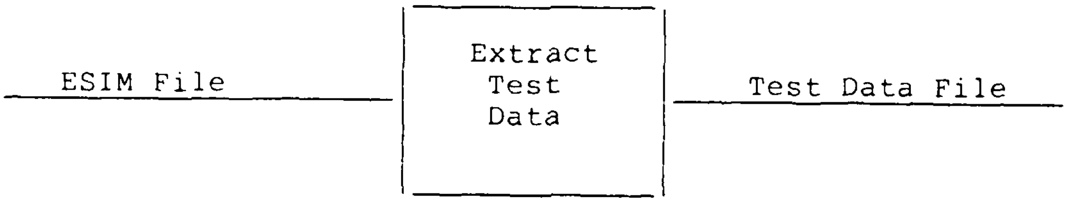
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Extract Test Data

PROCESS ID NUMBER: A1

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans ESIM-file for pertinent test data. Segregates it from other information and restructure this test data (available in node form) to test vectors and stores in a "new file".

INPUT DATA FLOW: ESIM File (VAX format)

OUTPUT DATA FLOW: Test Data File (LSI-11 format)

REFERENCE DIAGRAM: A0

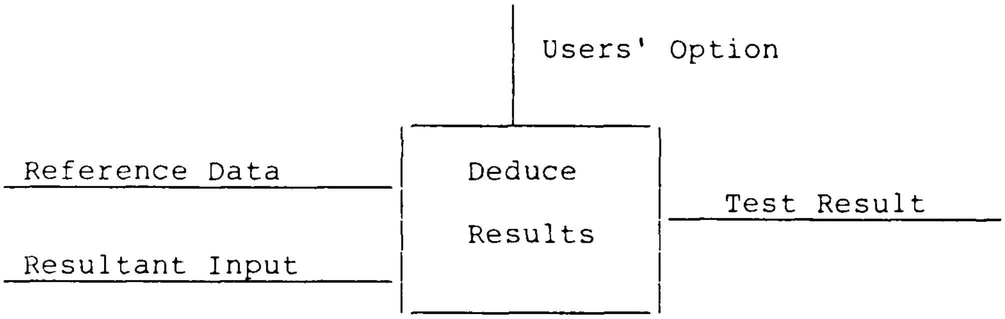
ADDITIONAL COMMENTS: The data format of "new file" the file in which restructures test data is stored, is changed from VAX-system to LSI-11 microcomputer data format and this file is transferred onto 8" floppy disk by system library routines.

PROCESS DEFINITION

PROCESS NAME: Deduce Results

PROCESS ID NUMBER: A4

PROCESS PICTURE:



PROCESS DESCRIPTION: This process translates the resultant output from IC tester pins domain to IC pin number domain and compares it with reference data for any non-conformity. This process generates test results from successful/unsuccessful completion of a test and pre-stored messages. This function also directs test results to terminal/disk file as opted by user.

INPUT DATA FLOW: Reference Data
Resultant Output

OUTPUT DATA FLOW: Test Results

REFERENCE DIAGRAM: A0

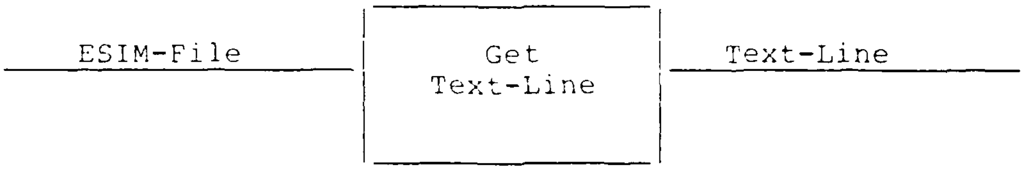
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Get Text-Line

PROCESS ID NUMBER: A111

PROCESS PICTURE:



PROCESS DESCRIPTION: This process reads a given ESIM-file line by line until end of file is reached. This process is implemented by a system library routine (fgets).

INPUT DATA FLOW: ESIM File

OUTPUT DATA FLOW: Text-Line

REFERENCE DIAGRAM: A11

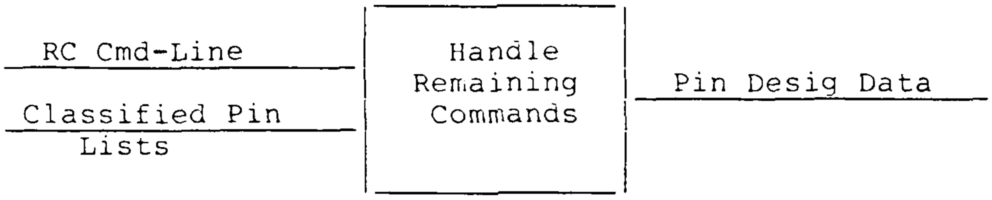
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Handle Remaining Commands

PROCESS ID NUMBER: A123

PROCESS PICTURE:



PROCESS DESCRIPTION: This process validates the "h or l" commands by making a check that designated output/clock pins are not driven to a "high/low" voltage status. This process changes the status of a valid pin to high/low as per received command.

INPUT DATA FLOW: RC Cmd-Line

Classified Pin Lists

OUTPUT DATA FLOW: Pin Desig Data

REFERENCE DIAGRAM: A12

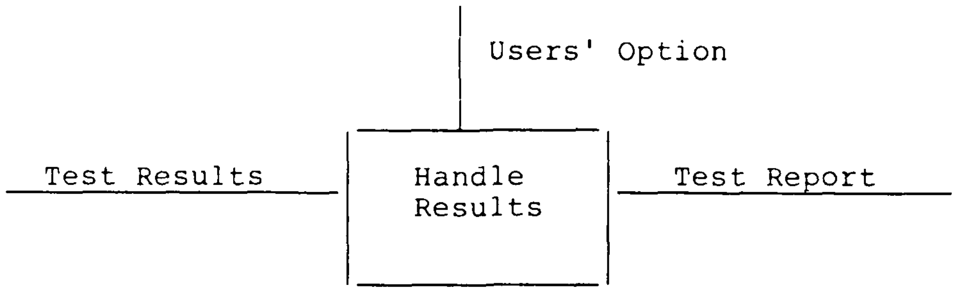
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Handle Results

PROCESS ID NUMBER: A42

PROCESS PICTURE:



PROCESS DESCRIPTION: This process maps a test result into a test report from pre-stored messages. This process as opted by user, allows display of test reports as terminal or storages of test reports in an external file.

INPUT DATA FLOW: Test Results

OUTPUT DATA FLOW: Test Report

REFERENCE DIAGRAM: A4

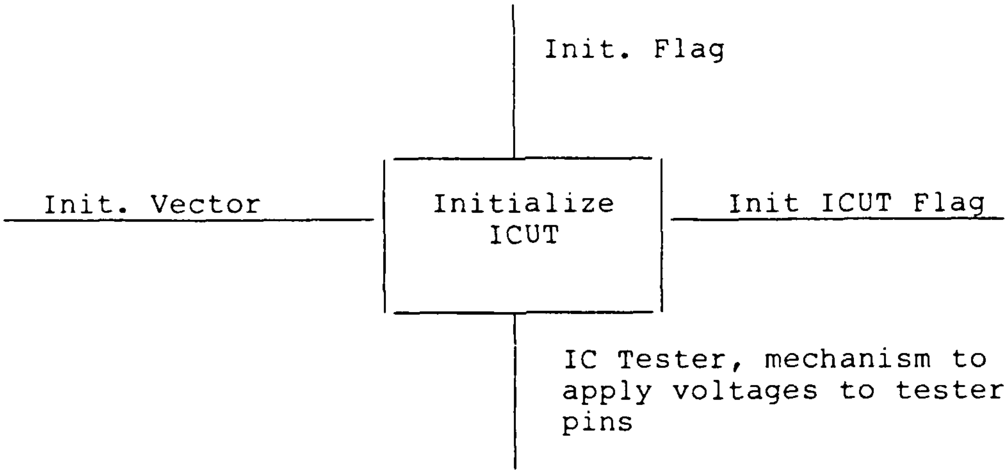
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Initialize ICUT

PROCESS ID NUMBER: A322

PROCESS PICTURE:



PROCESS DESCRIPTION: This process simulates ICUT, with users' supplied data or test data from test data file, through a pre-determined number of clock cycles to force the status of output pins to a steady state value.

INPUT DATA FLOW: Init. Vector

OUTPUT DATA FLOW: Init.ICUT flag

REFERENCE DIAGRAM: A32

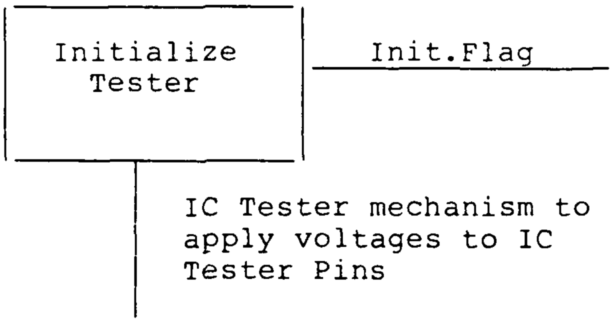
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Initialize Tester

PROCESS ID NUMBER: A321

PROCESS PICTURE:



PROCESS DESCRIPTION: This process physically applies "Ground/Power" voltages to pertinent pins of IC tester to force out all pins of IC tester of any ambiguous logic state.

INPUT DATA FLOW: None

OUTPUT DATA FLOW: Initialization (init) flag

REFERENCE DIAGRAM: A32

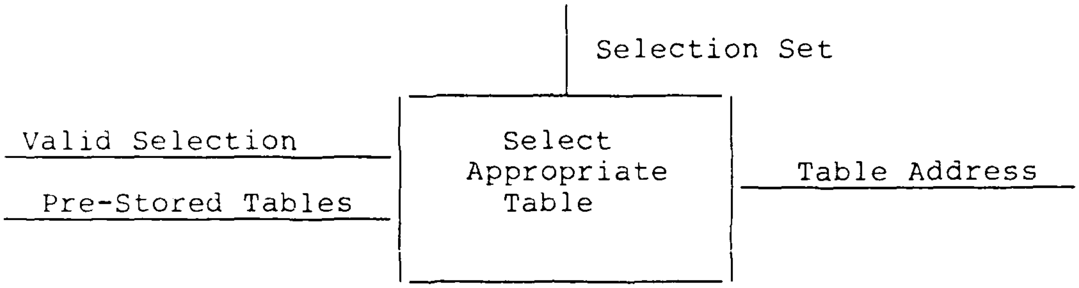
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Locate Appropriate Table

PROCESS ID NUMBER: A2314

PROCESS PICTURE:



PROCESS DESCRIPTION: This process correlates the valid selection with one of the pre-stored tables. this correspondence has been set aprior by the programmer.

INPUT DATA FLOW: Valid Selection

Pre-Stored Tables

OUTPUT DATA FLOW: Table Address

REFERENCE DIAGRAM: A231

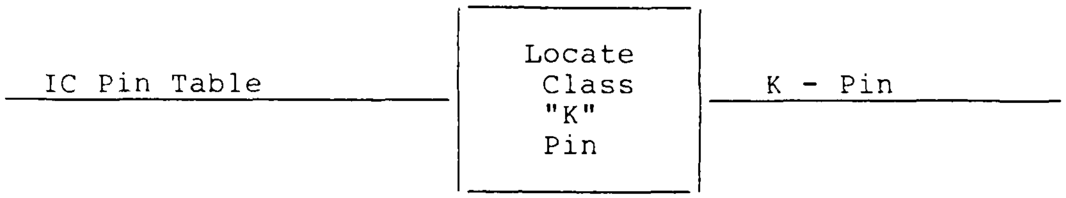
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Locate Class "K" Pin

PROCESS ID NUMBER: A2341

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans the class-field of IC pin table and points out an element whose class matches with "K".

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: K - Pin

REFERENCE DIAGRAM: A234

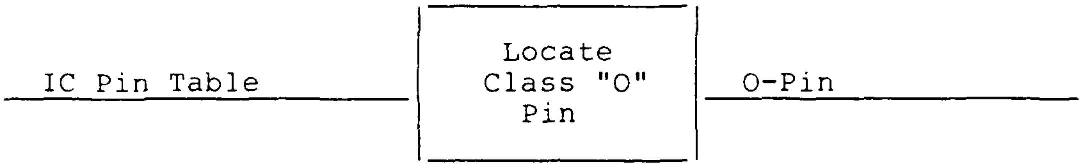
ADDITIONAL COMMENTS: This process is repeated to scan the whole IC pin table for "K" class (clock) pins.

PROCESS DEFINITION

PROCESS NAME: Locate Class "O" Pin

PROCESS ID NUMBER: A2361

PROCESS PICTURE:



PROCESS DESCRIPTION: this process scans the class-field of IC pin table and points out an element whose class matches with "O".

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: O-Pin

REFERENCE DIAGRAM: A236

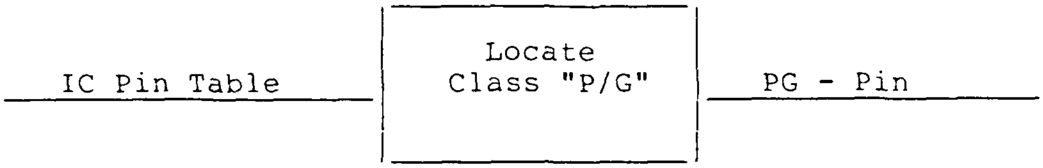
ADDITIONAL COMMENTS: This process is repeated to scan the whole IC pin table for "O" class (output) pins.

PROCESS DEFINITION

PROCESS NAME: Locate Class "P/G" Pin

PROCESS ID NUMBER: A2351

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans the class-field of IC pin table and points out an element whose class matches with "P/G".

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: PG-Pin

REFERENCE DIAGRAM: A23

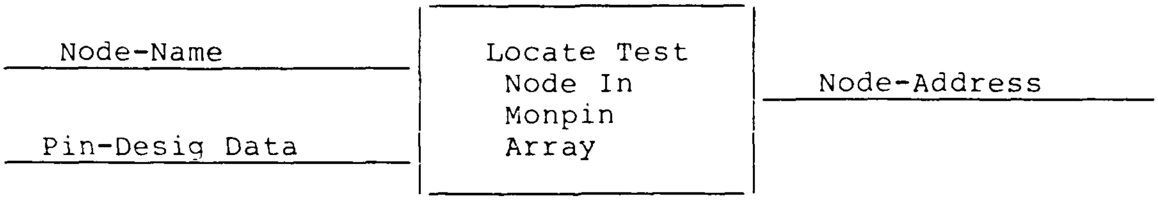
ADDITIONAL COMMENTS: This process is repeated to scan the whole IC pin table for all power/ground (P/G) class pins.

PROCESS DEFINITION

PROCESS NAME: Locate Test Node in Monpin Array

PROCESS ID NUMBER: A1312

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans the monpin array with "node name". If not found, it scans the array of input pins to find the match and returns the location (element number) of the matched pin.

INPUT DATA FLOW: Node-Name

Pin-Desig Data

- Monpin Array
- Inpin Array
- Clkpin Array
- Outpin Array

OUTPUT DATA FLOW: Node-Address

REFERENCE DIAGRAM: A131

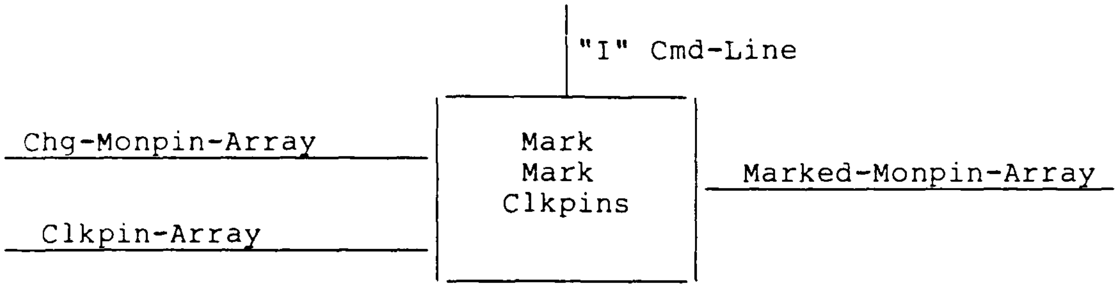
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Mark Clk-pins

PROCESS ID NUMBER: A12242

PROCESS PICTURE:



PROCESS DESCRIPTION: This process takes all elements of clkpin-array (array formed up by sequence of clock pins) one by one and matches each with all elements of array of monitored pins (monpin-array) and generates a cross-reference between elements of both arrays.

INPUT DATA FLOW: Chg-Monpin-Array

Clkpin-Array

OUTPUT DATA FLOW: Marked-Monpin-Array

REFERENCE DIAGRAM: A1224

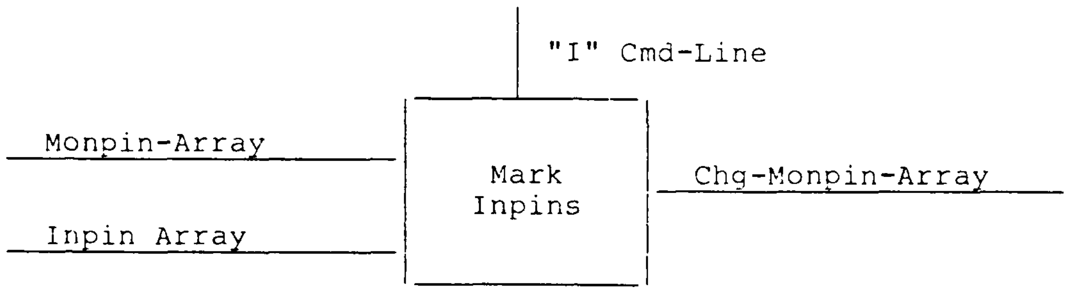
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Mark Inpins

PROCESS ID NUMBER: A12241

PROCESS PICTURE:



PROCESS DESCRIPTION: This process takes all elements of inpin-array (array formed up by sequence of input pins) one by one and matches each with all elements of array of monitored pins (monpin-array) and generates a cross-reference between elements of both arrays.

INPUT DATA FLOW: Monpin Array
Inpin Array

OUTPUT DATA FLOW: Chg-Monpin-Array

REFERENCE DIAGRAM: A1224

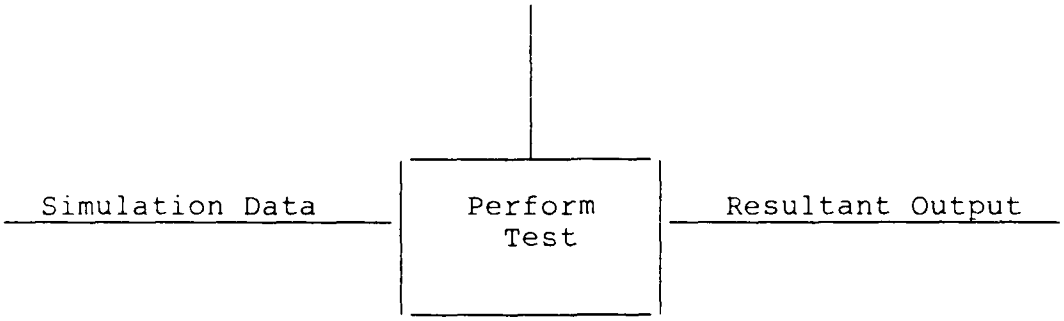
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Perform Test

PROCESS ID NUMBER: A32

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, initializes the IC tester, and ICUT before proceeding with actual test. It also applies test vector to IC tester to simulate ICUT and samples the output of IC under test for resultant output.

INPUT DATA FLOW: Simulation Data

OUTPUT DATA FLOW: Resultant Output

REFERENCE DIAGRAM: A3

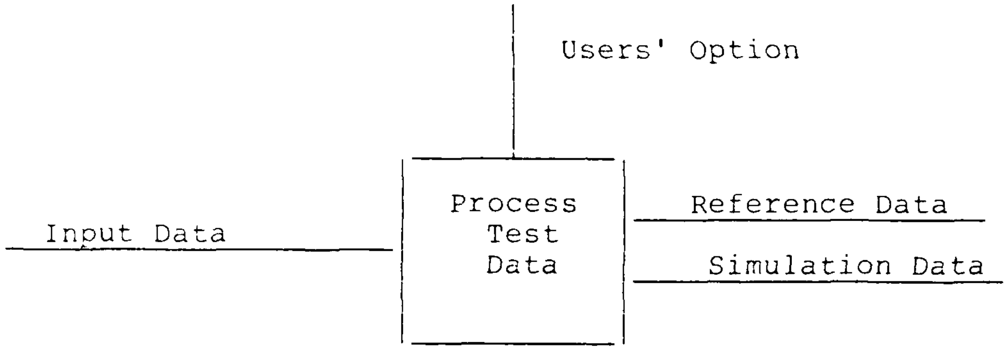
ADDITIONAL COMMENTS: Read description of Node A3 also (Page A-34).

PROCESS DEFINITION

PROCESS NAME: Process Test Data

PROCESS ID NUMBER: A31

PROCESS PICTURE:



PROCESS DESCRIPTION: This process establishes correspondence between test data vectors and pins of IC tester which are to be excited to effect simulation of ICUT. This process also changes the data format of input data into "SIEVE" format as it is required for operation of Stanford IC tester.

INPUT DATA FLOW: Input Data

OUTPUT DATA FLOW: Reference Data

Simulation Data

REFERENCE DIAGRAM: A3

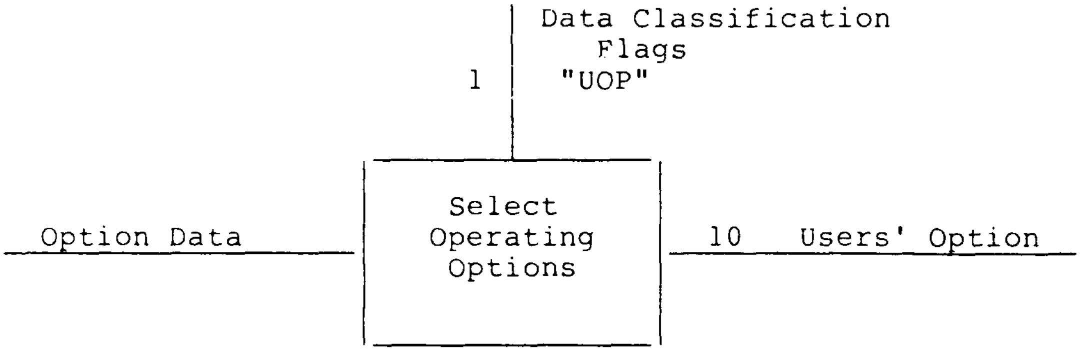
ADDITIONAL COMMENTS: Read description of Node A3 also (Page A-34).

PROCESS DEFINITION

PROCESS NAME: Select Operating Options

PROCESS ID NUMBER: A23

PROCESS PICTURE:



PROCESS DESCRIPTION: This proccss sets ten flags to execute the program in a manner, seiected by user, from system menues.

INPUT DATA FLOW: Option Data

OUTPUT DATA FLOW: Users' Option

REFERENCE DIAGRAM: A2

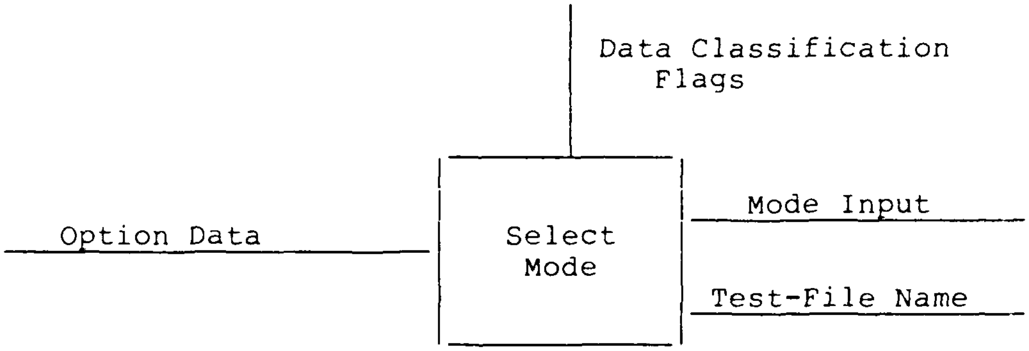
ADDITIONAL COMMENTS: Details of ten users' option and associated flags are included in description of Node A23.

PROCESS DEFINITION

PROCESS NAME: Select Mode

PROCESS ID NUMBER: A231

PROCESS PICTURE:



PROCESS DESCRIPTION: This process in response to system prompt receives "Auto/Manual" as preferred mode of operation. In case of "Auto-Mode" selection, user is asked to input name of respective test data file.

INPUT DATA FLOW: Option Data

- IC Nomenclature
- Mode of Operation
- Name of Test File

OUTPUT DATA FLOW: Mode Output

Test-File Name

REFERENCE DIAGRAM: A23

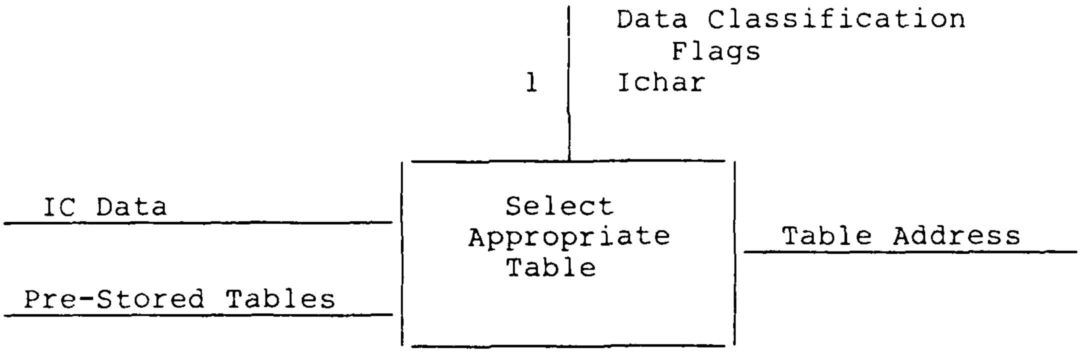
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Select Appropriate Table

PROCESS ID NUMBER: A241

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, selects one of the five pre-stored tables, on receiving size and pin information of an ICUT. The pre-stored table contains data to cross reference the ICUT pins and pins of IC tester.

INPUT DATA FLOW: IC Data

Pre-Stored Tables

OUTPUT DATA FLOW: Table Address

REFERENCE DIAGRAM: A24

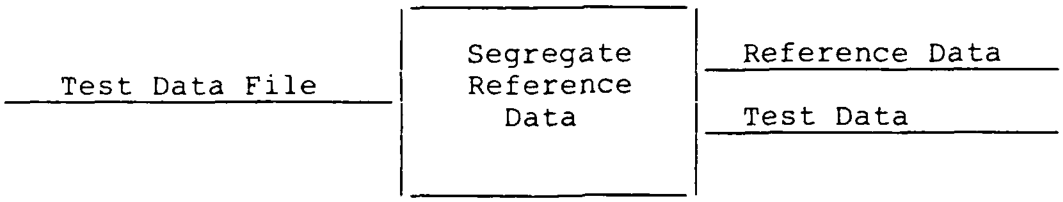
ADDITIONAL COMMENTS: Details of pre-stored tables is included in description of Node A24.

PROCESS DEFINITION

PROCESS NAME: Segregate Reference Data

PROCESS ID NUMBER: A311

PROCESS PICTURE:



PROCESS DESCRIPTION: This process reads in data form test data file (restructured ESIM file), separates out test data and expected output (reference data) for that particular simulation.

INPUT DATA FLOW: Test Data File (Restructured ESIM File)

OUTPUT DATA FLOW: Reference Data

Test Data

REFERENCE DIAGRAM: A31

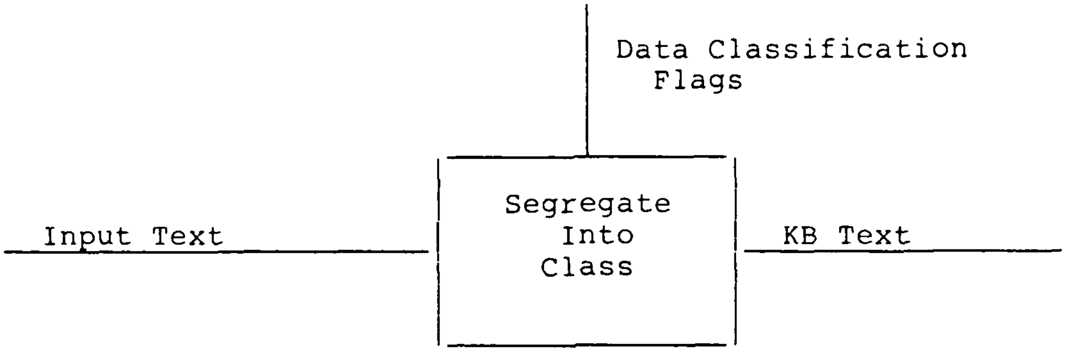
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Segregate into Class

PROCESS ID NUMBER: A223

PROCESS PICTURE:



PROCESS DESCRIPTION: This process segregates all the input text received in response to system prompts in three broad categories of IC, Option & Test Data. The combination of all three data categories is termed as KB text.

INPUT DATA FLOW: Input Text

OUTPUT DATA FLOW: KB Text

- IC Data
- Option Data
- Text Data

REFERENCE DIAGRAM: A22

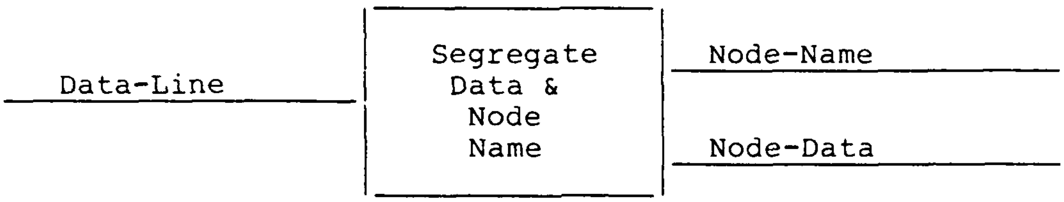
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Segregate Data & Node Name

PROCESS ID NUMBER: A1311

PROCESS PICTURE:



PROCESS DESCRIPTION: This process partitions a given data-line into node name and associated node data by scanning the whole line and sensing the presence of colon mark ":".

INPUT DATA FLOW: Data-Line

OUTPUT DATA FLOW: Node-Name
Node-Data

REFERENCE DIAGRAM: A131

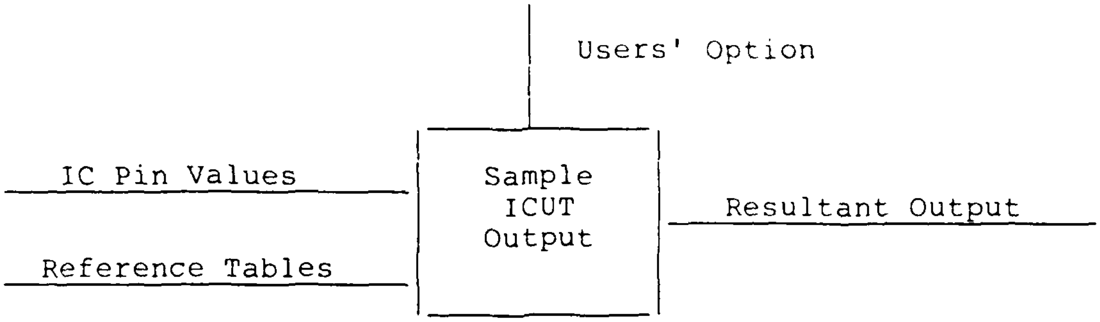
ADDITIONAL COMMENTS: Physical characteristics/details of a data-line in an ESIM file are shown on Page F-6.

PROCESS DEFINITION

PROCESS NAME: Sample ICUT Output

PROCESS ID NUMBER: A324

PROCESS PICTURE:



PROCESS DESCRIPTION: This process samples and stores the status of all pins of ICUT after it has been activated.

INPUT DATA FLOW: IC Pin Values

OUTPUT DATA FLOW: Resultant Output

REFERENCE DIAGRAM: A32

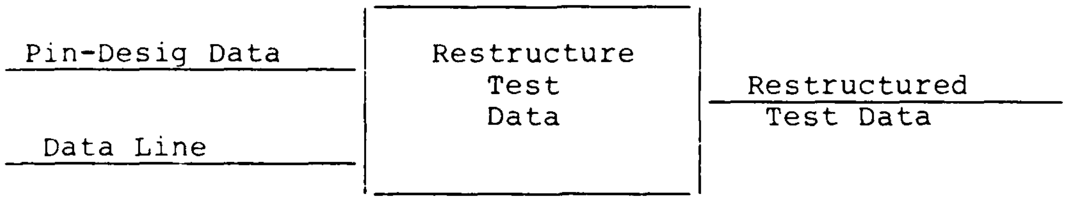
ADDITIONAL COMMENTS: The sampled value of ICUT pins is available to other function for any processing.

PROCESS DEFINITION

PROCESS NAME: Restructure Test Data

PROCESS ID NUMBER: A13

PROCESS PICTURE:



PROCESS DESCRIPTION: This process interprets a given data line and stores all the test data pertaining to a given node in its respective buffer. This process translates the test data from node form to test vectors and stores it in an external file whose name is provided by user.

INPUT DATA FLOW: Pin-Desig Data

Data-Line

OUTPUT DATA FLOW: Restructured Test Data

REFERENCE DIAGRAM: A1

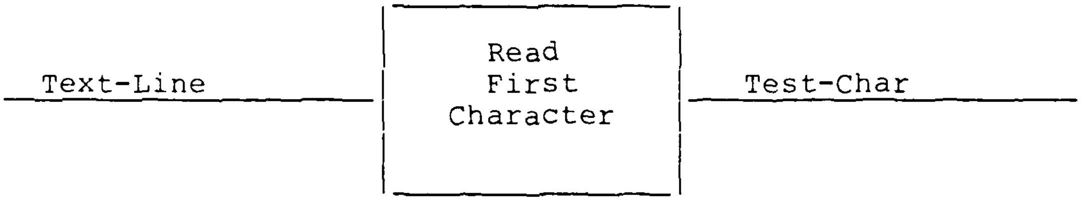
ADDITIONAL COMMENTS:

PROCESS DEFINITION

PROCESS NAME: Read First Character

PROCESS ID NUMBER: A1121

PROCESS PICTURE:



PROCESS DESCRIPTION: This process on receiving a text-line as input, considers it as an array of element and access its first element, which is termed as test-char (acter).

INPUT DATA FLOW: Text-Line

OUTPUT DATA fLOW: Test-Char

REFERENCE DIAGRAM: A112

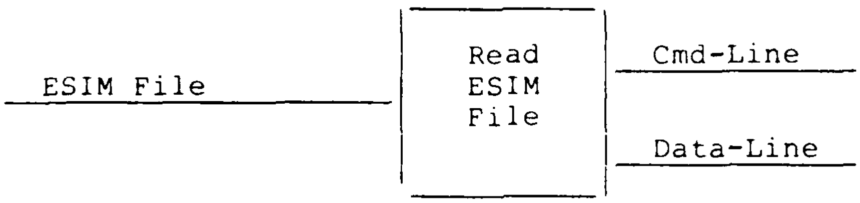
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Read ESIM-File

PROCESS ID NUMBER: All

PROCESS PICTURE:



PROCESS DESCRIPTION: This process leads in given ESIM file from the system memory line by line and classifies each incoming line to be command-line or data-line.

INPUT DATA FLOW: ESIM-File

OUTPUT DATA FLOW: Cmd-Line

Data-Line

REFERENCE DIAGRAM: Al

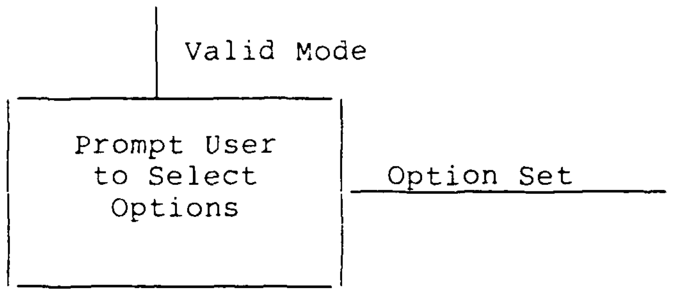
ADDITIONAL COMMENTS: A bried explanation of all commands encountered in ESIM files and a sample of typical file is attached as Appendix "D".

PROCESS DEFINITION

PROCESS NAME: Prompt User to Select Option

PROCESS ID NUMBER: A2231

PROCESS PICTURE:



PROCESS DESCRIPTION: This process on deciding the validity of node Auto or Manual, offers from selected mode a set of sub-options to the user in a menu. The details of sub-options are narrated in description of Node A22.

INPUT DATA FLOW: -

OUTPUT DATA FLOW: Option Set

REFERENCE DIAGRAM: A223

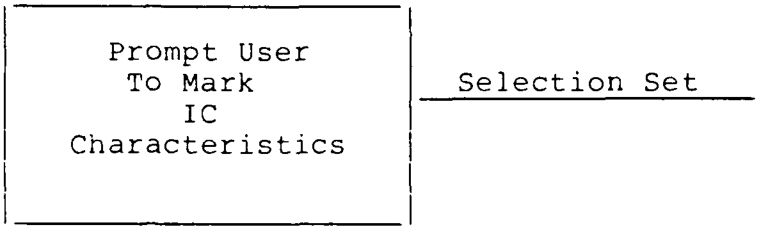
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Prompt User to Mark IC-Characteristics

PROCESS ID NUMBER: A2311

PROCESS PICTURE:



PROCESS DESCRIPTION: This process activated in sequence after a valid users' option has been made, offers a menu containing various combinations of IC Characteristics [i.e., A) 20 pin dual in line B) 40 pin dual in line, C) 40 pin square flat pack, etc.] The user is asked to make one selection which exactly matches the physical description of IC to be tested.

INPUT DATA FLOW: -

OUTPUT DATA FLOW: Selection Set

REFERENCE DIAGRAM: A231

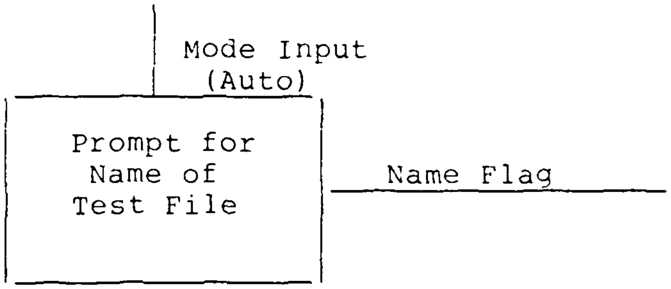
ADDITIONAL COMMENTS: Tables containing data are stored aprior by the programmer which set correspondence between IC pins and tester pins. If no physical description matches that of IC to be tested, a new table will have to be setup for testing that IC.

PROCESS DEFINTION

PROCESS NAME: Prompt for Name of Test File

PROCESS ID NUMBER: A2213

PROCESS PICTURE:



PROCESS DESCRIPTION: This process in case of "Auto-Mode" selection asks user to input name of respective test file and sets a flag to process next input (name of test file) from keyboard.

INPUT DATA FLOW: -

OUTPUT DATA FLOW: Name Flag

REFERENCE DIAGRAM: A221

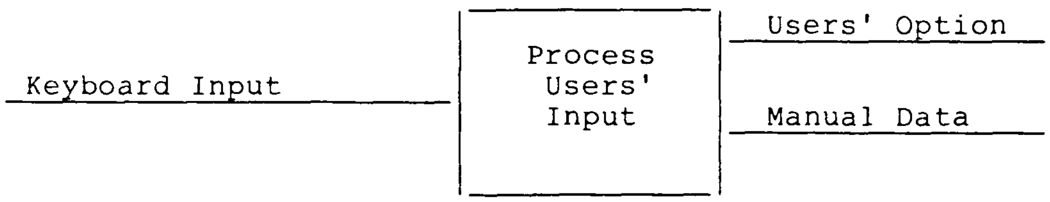
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Process Users' Input

PROCESS ID NUMBER: A2

PROCESS PICTURE:



PROCESS DESCRIPTION: This process prompts the user to input data/commands then the keyboard in response to the menu offered to the user. This process sets various option flags for mode of operation and receives IC characteristic data to setup stage to conduct tests. This process also validates the input test data in manual mode of operation.

INPUT DATA FLOW: Keyboard Input

OUTPUT DATA FLOW: Manual Data (Test data for manual mode of operation)

Users' Option (Please read description of nodes A2 & A23 for details)

REFERENCE DIAGRAM: A0

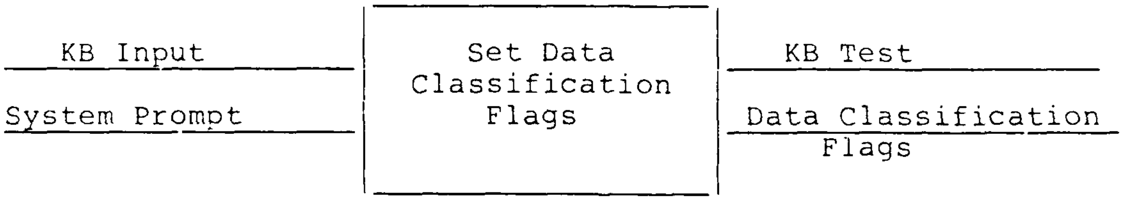
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Set Data-Classification Flags

PROCESS ID NUMBER: A212

PROCESS PICTURE:



PROCESS DESCRIPTION: This process sets one of three data classification flagsi.e., idflag, opflag, and tdflag from system prompt to classify the keyboard input in three broad categories of IC data, Option Data and Test Data for further processing.

INPUT DATA FLOW: KB Input
System Prompt

OUTPUT DATA FLOW: KB Text
Data Classification Flag

REFERENCE DIAGRAM: A21

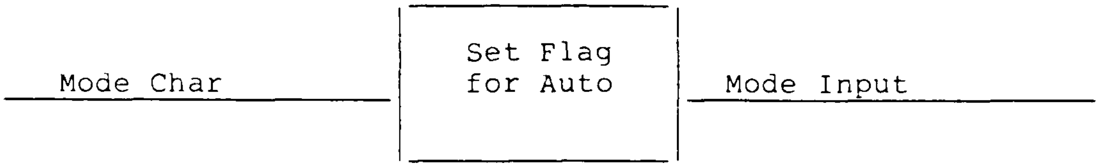
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Set Flag for Auto

PROCESS ID NUMBER: A2212

PROCESS PICTURE:



PROCESS DESCRIPTION: This process sets respective flags to be true for"Auto or Manual" mode of operation on receiving "mode character" from keyboard.

INPUT DATA FLOW: Mode Char

OUTPUT DATA FLOW: Mode Input

REFERENCE DIAGRAM: A221

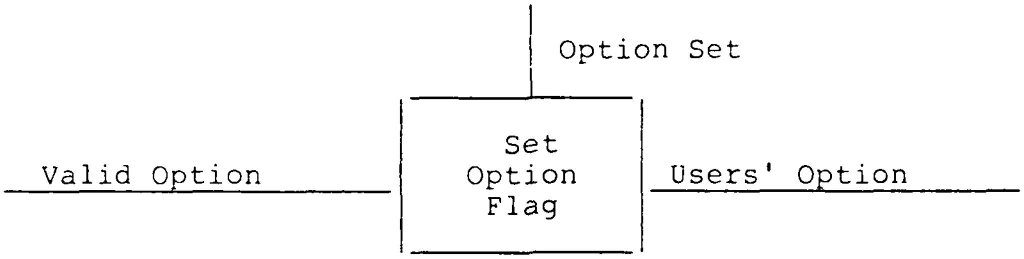
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Set Option Flag

PROCESS ID NUMBER: A2234

PROCESS PICTURE:



PROCESS DESCRIPTION: This process on receiving a valid option, sets the respective flag (nomenclature of all flags listed in description of Node A22) to be true.

INPUT DATA FLOW: Valid Option

OUTPUT DATA FLOW: Users' Option

REFERENCE DIAGRAM: A223

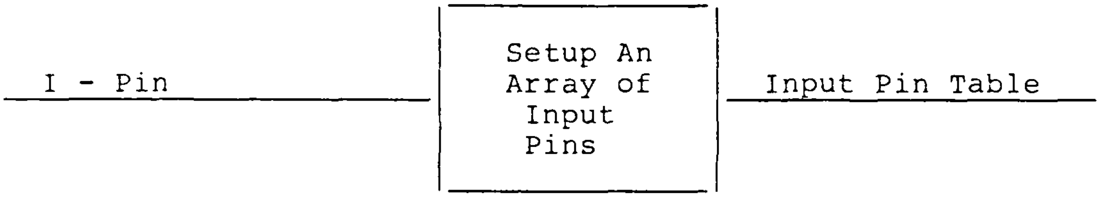
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Setup an Array of Input Pins

PROCESS ID NUMBER: A2322

PROCESS PICTURE:



PROCESS DESCRIPTION: This process adds each "I" class elements of IC Pin Table to a separate array which has been defined aprior by the programmer.

INPUT DATA FLOW: I-Pin

OUTPUT DATA FLOW: Input Pin Table

REFERENCE DIAGRAM: A233

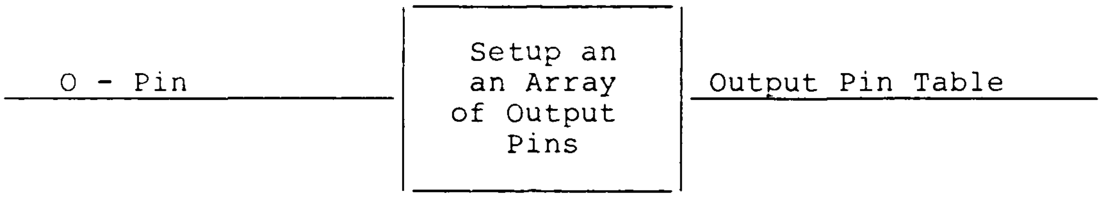
ADDITIONAL COMMENTS: This process is repeated to add all encountered elements by Node A2331.

PROCESS DEFINITION

PROCESS NAME: Setup an Array of Output Pins

PROCESS ID NUMBER: A2362

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, adds each "O" class elements of IC pin table to a separate array, which has been defined aprior by the programmer.

INPUT DATA FLOW: O-Pin

OUTPUT DATA FLOW: Output Pin Table

REFERENCE DIAGRAM: A236

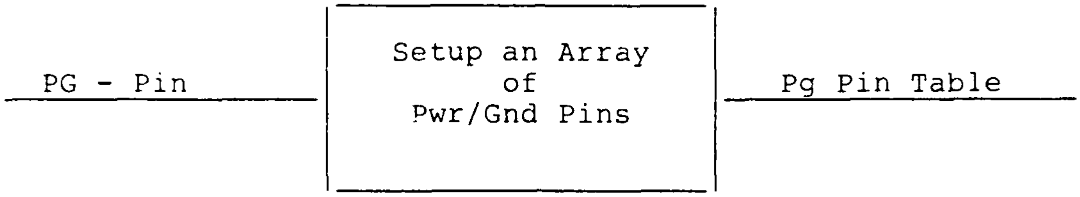
ADDITIONAL COMMENTS: This process is repeated to add all encountered elements by Node A236.

PROCESS DEFINITION

PROCESS NAME: Setup an Array of Pwr/Gnd Pins

PROCESS ID NUMBER: A2352

PROCESS PICTURE:



PROCESS DESCRIPTION: This process adds each "P/G" class elements of IC Pin Table to a separate array, which has been defined aprior by the programmer.

INPUT DATA FLOW: PG-Pin

OUTPUT DATA FLOW: PG Pin Table

REFERENCE DIAGRAM: A235

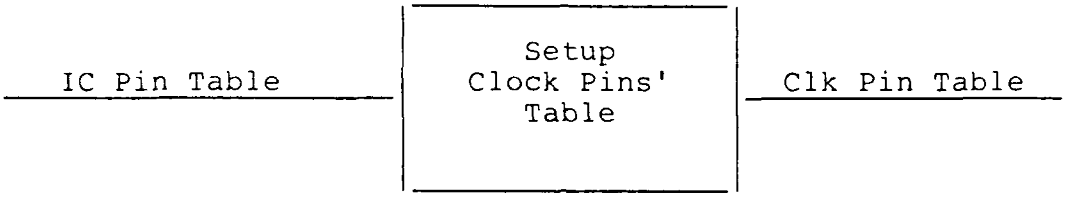
ADDITIONAL COMMENTS: This process is repeated to add all encountered elements by Node A2351.

PROCESS DEFINITION

PROCESS NAME: Setup Clock Pins' Table

PROCESS ID NUMBER: A244

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, scans the reference table after it has been completely filled-in, and sets up a separate table of pins marked as "clock" pins.

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: Clk Pin Table

REFERENCE DIAGRAM: A24

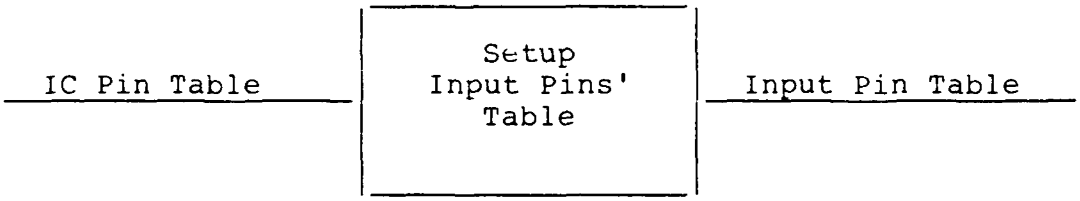
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Setup Input Pins' Table

PROCESS ID NUMBER: A243

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans the reference table after it has been completely filled-in, and sets up a separate table of pins - marked as "Input" Pins.

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: Input Pin Table

REFERENCE DIAGRAM: A24

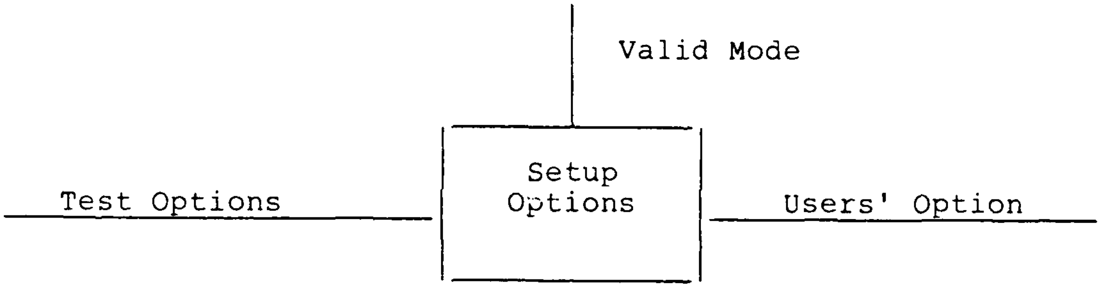
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Setup Options

PROCESS ID NUMBER: A233

PROCESS PICTURE:



PROCESS DESCRIPTION: This process sets ten different flags to be "true/false", to execute the program in a manner selected by user from system menues.

INPUT DATA FLOW: Test Options

OUTPUT DATA FLOW: Users' Option (10 flags)

REFERENCE DIAGRAM: A22

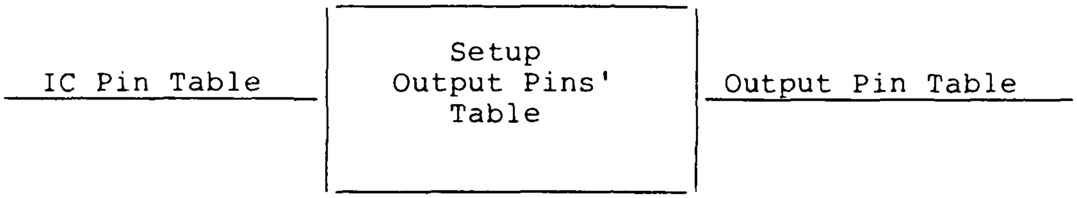
ADDITIONAL COMMENTS: The details about users' options and associated flags are included in description of Node A23.

PROCESS DEFINITION

PROCESS NAME: Setup Output Pins' Table

PROCESS ID NUMBER: A246

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, scans the reference table after it has been completely fillin and sets up a separate table of pins marked as "output" pins.

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: Output Pin Table

REFERENCE DIAGRAM: A24

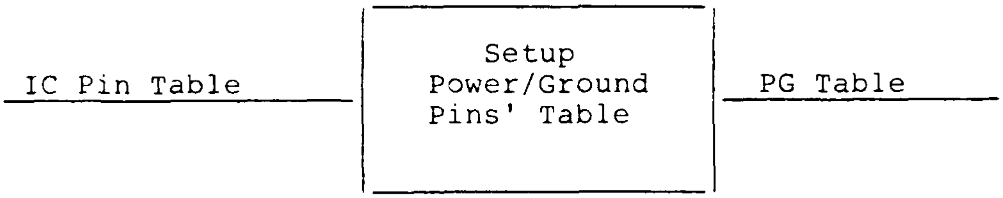
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Setup Power/Ground Pins Table

PROCESS ID NUMBER: A245

PROCESS PICTURE:



PROCESS DESCRIPTION: This process scans the reference table after it has been completely filled in and sets up a separate table of pins marked as "Power/Ground" pins.

INPUT DATA FLOW: IC Pin Table

OUTPUT DATA FLOW: PG Table

REFERENCE DIAGRAM: A24

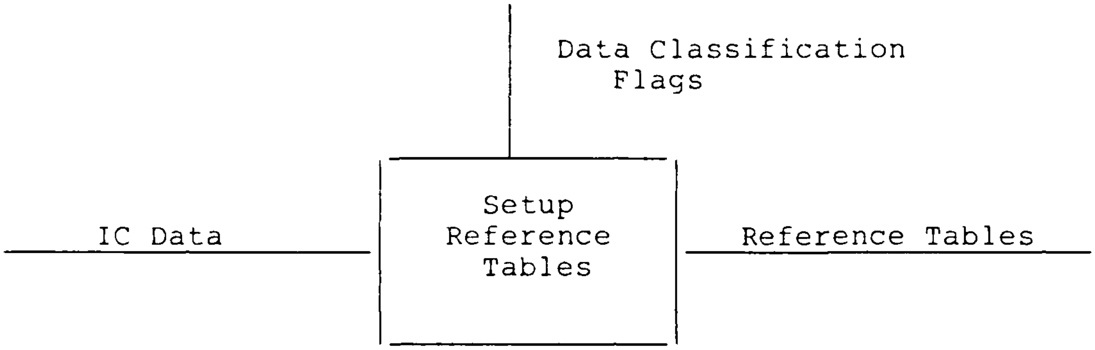
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Setup Reference Tables

PROCESS ID NUMBER: A24

PROCESS PICTURE:



PROCESS DESCRIPTION: This process selects one of the five pre-stored tables depending on physical characteristics of an IC to be tested. The selected tables contain cross-reference data between ICUT pins and IC tester pins. The selected table is filled by prompting user to input information regarding pin names and pin numbers of an ICUT.

INPUT DATA FLOW: IC Data

OUTPUT DATA FLOW: Reference Tables

REFERENCE DIAGRAM: A2

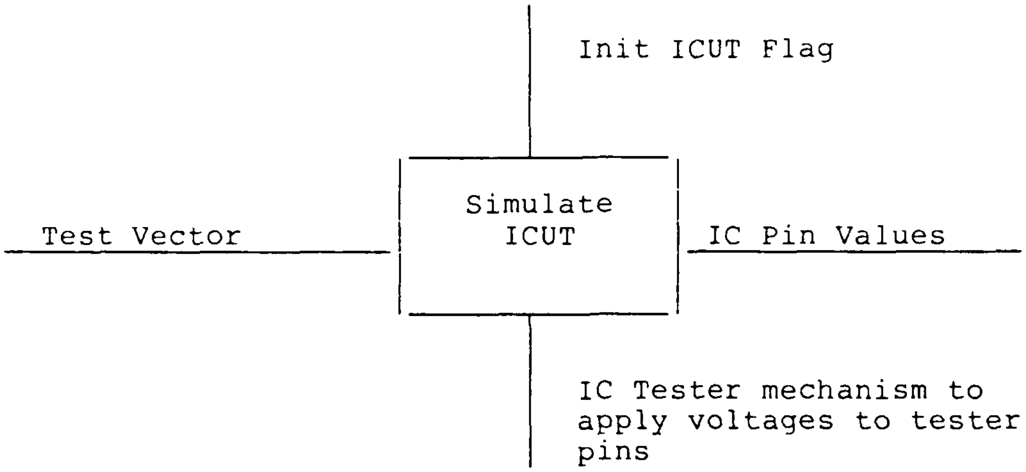
ADDITIONAL COMMENTS: Details about pre-stored tables are included in descriptions of Node A2 and A24.

PROCESS DEFINITION

PROCESS NAME: Simulate ICUT

PROCESS ID NUMBER: A323

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, after being supplied with a test vector and a flag to proceed, applies corresponding voltages to effect simulation.

INPUT DATA FLOW: Test Vector

OUTPUT DATA FLOW: IC Pin Values

REFERENCE DIAGRAM: A32

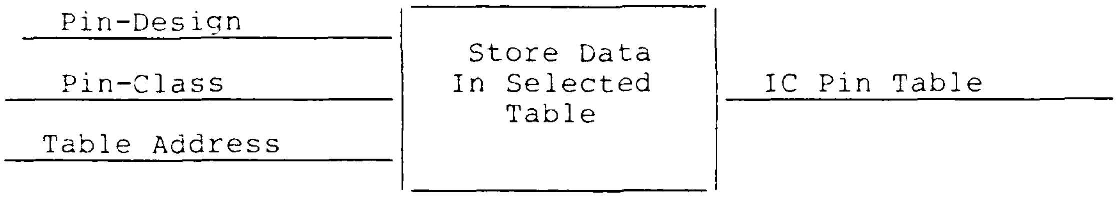
ADDITIONAL COMMENTS: This process also translates the clocking sequence into voltage variation at designated clock-pin.

PROCESS DEFINITION

PROCESS NAME: Store Data in Selected Table

PROCESS ID NUMBER: A2323

PROCESS PICTURE:



PROCESS DESCRIPTION: This process stores the designated names and class of all pins. It stores this information in respective fields of selected table for future reference.

INPUT DATA FLOW: Pin-Desig

Pin-Class

Table Address

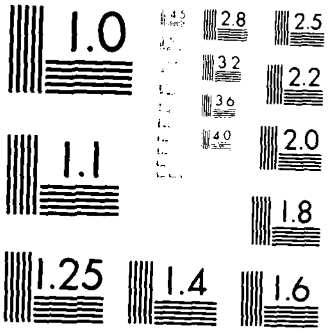
OUTPUT DATA FLOW: IC Pin Table

REFERENCE DIAGRAM: A232

ADDITIONAL COMMENTS: None

AD-A151 961 SYSTEM DESIGN OF AUTOMATED VLSI (VERY LARGE SCALE 4/4
INTEGRATED) TEST STATIO..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. S TARIQ
UNCLASSIFIED DEC 84 AFIT/GE/EE/84D-27 F/G 9/5 NL

END
FILMED
DTIC

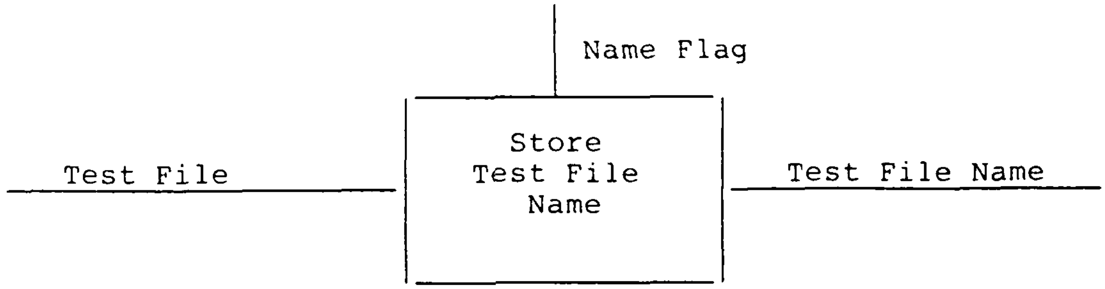

1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

PROCESS DEFINITION

PROCESS NAME: Store Test File Name

PROCESS ID NUMBER: A2214

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, activated by control flag "name flag" receives name of test file input then the keyboard in response to the system prompt and stores this name in a specific buffer for future reference.

INPUT DATA FLOW: Test File

OUTPUT DATA FLOW: Test File Name

REFERENCE DIAGRAM: A221

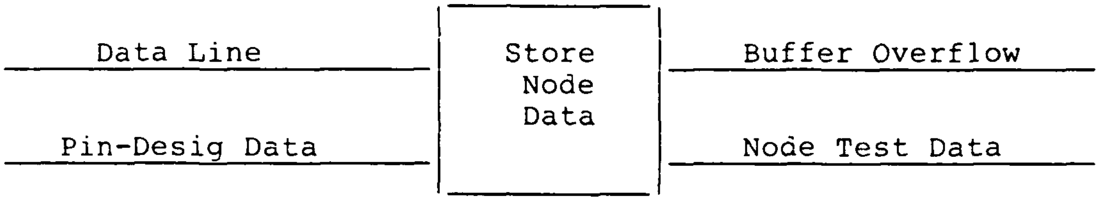
ADDITIONAL COMMENTS: This process, is readily implemented in "C" language by system library function (scanf (...)).

PROCESS DEFINITION

PROCESS NAME: Store Node Data

PROCESS ID NUMBER: A131

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, segregates node data in respective of a pin and stores it in its associated buffer. It generates an overflow signal if incoming data exceeds the remaining capacity of the buffer for a particular pin.

INPUT DATA FLOW: Data Line
Pin Desig Data

OUTPUT DATA FLOW: Buffer Over Flow
Node Test Data

REFERENCE DIAGRAM: A13

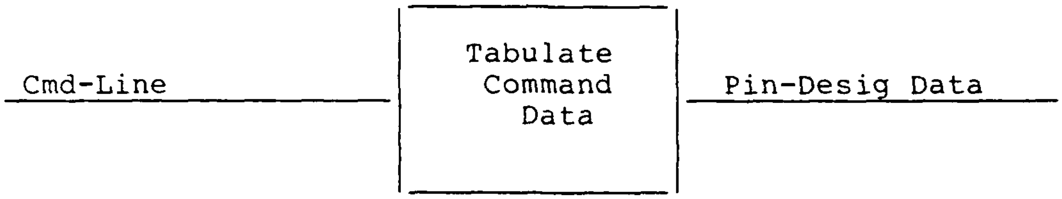
ADDITIONAL COMMENTS: An example of ESIM file showing typical format of a "data line" is included in Appendix "D".

PROCESS DEFINITION

PROCESS NAME: Tabulate Command Data

PROCESS ID NUMBER: A12

PROCESS PICTURE:



PROCESS DESCRIPTION: This process interprets the command in a given "Cmd-Line" to setup arrays to store names and related data in respective of all monitored pins. This process also generates an array of output pins and establishes reference between elements of input output and clk. pin arrays and master array containing all monitored pins.

INPUT DATA FLOW: Cmd-Line

OUTPUT DATA FLOW: Pin-Desig Data - Pointers to arrays describing all types of IC pins.

REFERENCE DIAGRAM: A1

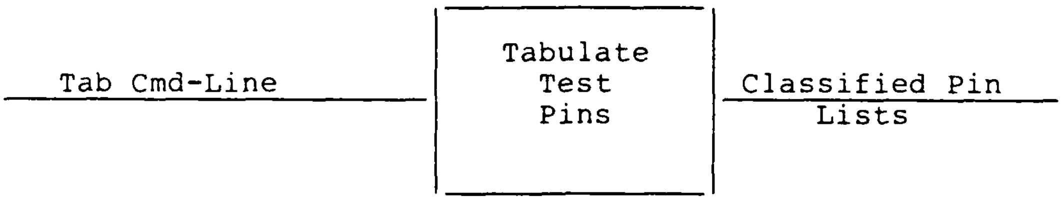
ADDITIONAL COMMENTS: Please read description of Node A12 for details.

PROCESS DEFINITION

PROCESS NAME: Tabulate Test Pins

PROCESS ID NUMBER: A122

PROCESS PICTURE:



PROCESS DESCRIPTION: This process generates arrays of monitored pins, input pins and clock pins on receiving a specific command. It establishes reference between above three arrays and generates an array of output pins. This process also stores related data for each pin in its respective buffers.

INPUT DATA FLOW: Tab Cmd-Line

OUTPUT DATA FLOW: Classified Pin Lists

- Array of Monitored Pins
- Array of Input Pins
- Array of Clock Pins
- Array of Output Pins

REFERENCE DIAGRAM: A12

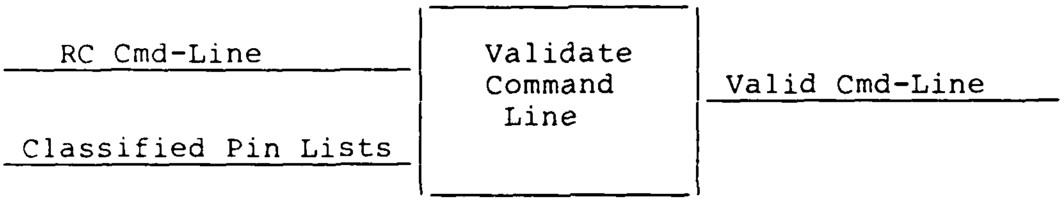
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Validate Command Line

PROCESS ID NUMBER: A1231

PROCESS PICTURE:



PROCESS DESCRIPTION: This process, on interpreting first character of incoming command-line to be "h or l" scans the arrays of output and clock pins to insure that any designated pin from this class is not forced to have "high/ low" status. It marks the command to be valid if no output/clock pin is effected by "h/l" command.

INPUT DATA FLOW: RC Cmd-Line

Classified Pin Lists

OUTPUT DATA FLOW: Valid Cmd-Line

REFERENCE DIAGRAM: A123

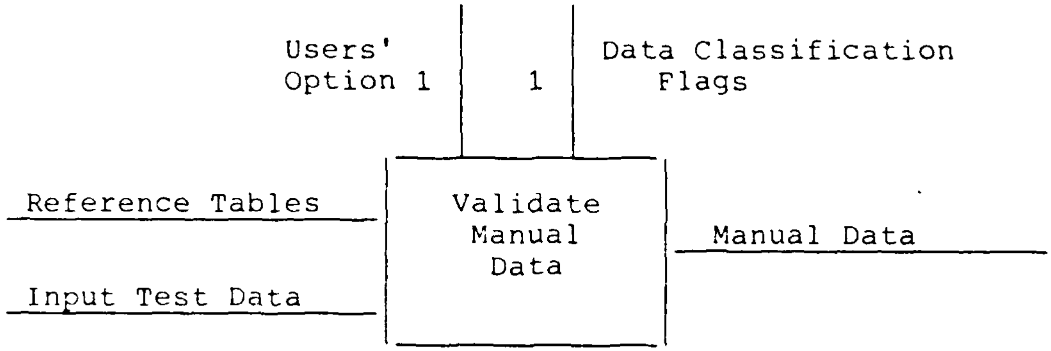
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Validate Manual Data

PROCESS ID NUMBER: A25

PROCESS PICTURE:



PROCESS DESCRIPTION: This process activated only during "Manual" node of operation checks to insure that input test data does not overlap any designated ouput clock or Power/ Ground Pins.

INPUT DATA FLOW: Reference Tabvles
Input Test Data

OUTPUT DATA FLOW: Manual Data

REFERENCE DIAGRAM: A2

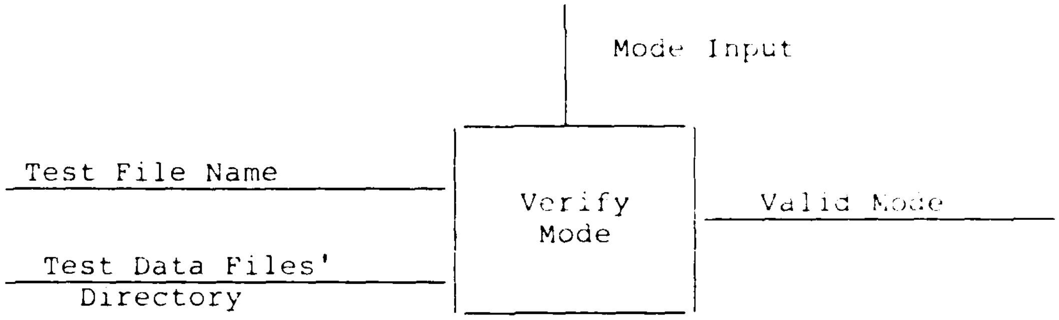
ADDITIONAL COMMENTS: None

PROCESS DEFINITION

PROCESS NAME: Verify Mode

PROCESS ID NUMBER: A232

PROCESS PICTURE:



PROCESS DESCRIPTION: This process checks if a particular test-data file is available for possible operation in case of auto mode selection. It generates an error, if test-data file for an ICUT is not available.

INPUT DATA FLOW: Test File Name

Test Data Files' Directory

OUTPUT DATA FLOW: Valid Mode

REFERENCE DIAGRAM: A23

ADDITIONAL COMMENTS: None

Appendix "H"

User Manual

ETD(1) UNIX Programmer's Manual ETD(1)

Name

ETD - Extract Test Data

Synopsis

Description

ETD is a program that extracts pertinent test data from a given ESIM file. Any ESIM file consists of simulator test runs for a particular VLSI circuit during its design phase. A typical format of ESIM file is shown on Page H-3. ETD segregates node data and converts it into vector form. A typical resultant output is shown on Page H-4.

To use ETD, you must input the name of ESIM file and name of file in which you want to store test vectors. The program will ask for the names of these files in interactive manner during its execution.

An example is shown below:

```
                    etd   <cr>
      ENTER NAME OF ESIM (TEST-DATA) FILE:
                 node-data  <cr>
            ENTER NAME OF STORAGE FILE:
                 warehouse  <cr>
```

If the "test data" file does not exist in the same directory, the program will quit after printing a message that "test data" file couldn't be opened for reading.

If "test-data" file actually exists, the program (ETD) would list the names of input, clock output pins and resturctured test vector, along with respective reference output vector in the "storage" file, as shown on Page H-4.

The column Nos. 1 and 2 of output (Page H-4) display input and output test vectors. The individual bits of these vectors correspond to test pins listed in inpins and outpins in the same order. The "Ø" in the 3rd column indicates the last of test vectors and "Ø" in the 4th column indicates a change in designation of test pins.

See Also

AFIT/GE/EE/84D-27   MS Thesis

Author

Saleem

Bugs

1. User has to input name of a valid ESIM file, otherwise ETD would produce some unintelligent output.

2. The last line of ESIM file must not be a command line or a data line; it should be a comment line.

Typical ESIM-File
(Data Format)

```
w clock serial-in word-mark w0 w1 w2 w3 to t1 t2 t3 t4 t5
V clock 01010101010101010101
V serial-in 00110011001100110011
V word-mark 00000011000000000011
V reset 11000000000000000000
I
R
456 transistors, 220 nodes (0 pulled up)
Initialization took 285 steps
t5=X t4=X t3=X t2=X t1=X t0=X w3=X w2=X w1=X w0=X word-
mark=X serial in=X clock=X
h inputs: Vdd gnd
l inputs: GND vdd reset
t5=0 t4=0 t3=0 t2=0 t1=0 t0=0 w3=0 w2=0 w1=0 w0=0 word-
mark=1 serial-in=1 clock=1
h inputs; Vdd gnd clock serial-in word-mark
l inputs: GND Vdd reset
)00110011001100110011:serial in
)00000011000000000011:word mark
)00000000000000000000:w0
)00000000000000000000:w1
)00000000000000000000:w2
)00000000000000000000:w3
)00000000000000000000:t0
)00000000000000000000:t1
)00000000000000000000:t2
)00000000000000000000:t3
)00000000000000000000:t4
)00000000000000000000:t5
t5=0 t4=0 t3=0 t2=0 t1=0 t0=0 w3=0 w2=0 w1=0 w0=0 word-
mark=1 serial-in=1 clock=1
h inputs: Vdd gnd clock serial-in word-mark
l inputs: GND vdd reset
456 transistors, 220 nodes (0 pulled up)
%
```

Typical Output
(Data Format)

```
 IMPIN     4     clock serial_in word_mark reset
 CLKPIN    0
 OUTPIN   10     w0 w1 w2 w3 t0 t1 t2 t3 t4 t5
>0001               0000000000          1          0
>1001               0000000000          1          0
>0100               0000000000          1          0
>1100               0000000000          1          0
>0000               0000000000          1          0
>1000               0000000000          1          0
>0110               0000000000          1          0
>1110               0000000000          1          0
>0000               0000000000          1          0
>1000               0000000000          1          0
>0100               0000000000          1          0
>1100               0000000000          1          0
>0000               0000000000          1          0
>1000               0000000000          1          0
>0100               0000000000          1          0
>1100               0000000000          1          0
>0000               0000000000          1          0
>1000               0000000000          1          0
>0110               0000000000          1          0
>1110               0000000000          0          0
```

Bibliography

1. Peter, Lawrence J. Software_Design:_Methods_& Techniques, New York: Yourdin Press, 1981.

2. Pressman, Roger S. Software_Engineering:__A Practitioner's_Approach, New York: McGraw-Hill Book Company, 1982.

3. Softech, Inc., "An Introduction to Structured Analysis and Design Technique" (Softech Document #9022-78) November 1976.

4. Myers, G. The_Art_of_Stoftware_Testing, New York: Wiley, 1979.

5. Kernighan, Brian W. & Ritchie, Dennis M. The_C Programming_Language, Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1978.

6. Kochan, Stephen G. Programming_in_C, New Jersey: Hayden Book Company, Inc. 1983.

7. Plum, Thomas. Learning_to_Program_in_C, Englewood Cliffs, New Jersey: Prentice-hall, Inc., 1983.

8. Peter, Lawrence J. "Software Representation and Composition Techniques", Proceedings_of_the_IEEE, Vol. 68. No. 9, September 1980.

9. Wirth, Niklaus. "Program_Development_by_Stepwise Refinement" Communications of the ACM, Vol. 14, No. 4, April 1971 (PP221-226).

10. Reifer, Donald J. & Trattner, Stephen. [illegible] Specification_Techniques:__A [illegible]

11. Ross, Douglas T. "Structured Analysis (SA): A Language for Communicating Ideas", [illegible] Vol. SE-3, No. 1, (pp. 16-34), January 1977.

12. Ross, Douglas T. and Schoman, K. E. "Structured Analysis for Requirement Definition", IEEE Transactions, Vol. SE-3, No. 1, (pp. 6-15), January 1977.

VITA

Sqn. Ldr. Saleem Tariq was born on 2nd November 1952 in Wah Cantt, Pakistan. He did his matriculation and F.Sc with distinction from PAF College, Sargodha in 1969 and 1971 respectively. He graduated from the College of Aeronautical Engineering, Karangi Creek (Karachi) in 1976 with a Bachelor of Engineering degree in Avionics. He was awarded a permanent commission in the Pakistan Air Force on his graduation and since then has been appointed to various duties as an Electronic Maintenance Officer. In June 1983, he was assigned to the School of Engineering, Air Force Institute of Technology to complete his master's degree in Electrical Engineering.

Permanent Address: 85-A
Lalarukh, Wah Cantt
Pakistan

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: Unclassified

1b RESTRICTIVE MARKINGS:

2a SECURITY CLASSIFICATION AUTHORITY:

2b DECLASSIFICATION/DOWNGRADING SCHEDULE:

3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.

4 PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/EE/84D-27

5 MONITORING ORGANIZATION REPORT NUMBER(S):

6a NAME OF PERFORMING ORGANIZATION: School of Engineering

6b OFFICE SYMBOL *(If applicable)*: AFIT/ENG

7a NAME OF MONITORING ORGANIZATION:

6c ADDRESS *(City, State and ZIP Code)*:
Air Force Institute of Technology
Wright Patterson Air Force Base
Ohio 45433

7b ADDRESS *(City, State and ZIP Code)*:

8a NAME OF FUNDING/SPONSORING ORGANIZATION: See Box 6a

8b OFFICE SYMBOL *(If applicable)*: AFIT/ENG

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:

8c ADDRESS *(City, State and ZIP Code)*: See Box 6c.

10 SOURCE OF FUNDING NOS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT NO

11 TITLE *(Include Security Classification)*: System Design of Automated VLSI Test Station and Implementation of Selected System Aspects (Unclassified)

12 PERSONAL AUTHOR(S): Sqn. Ldr. Saleem Tariq, Pakistan Air Force

13a TYPE OF REPORT: Master of Electrical Eng

13b TIME COVERED: FROM TO

14 DATE OF REPORT *(Yr., Mo., Day)*: 1984 December 2nd

15 PAGE COUNT: 2[illegible]

16 SUPPLEMENTARY NOTATION:

Approved for public release: IAW AFR 190-17.
[illegible]
Dean for [illegible] Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH

17 COSATI CODES

FIELD	GROUP	SUB. GR.
09	02	
	0[illegible]	

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*: VLSI Testing, Microcomputer, Computer Programs,

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Automated Test Station for VLSI [illegible] to ascertain correct functioning of a VLSI circuit. It is intended to test an Integrated Circuit (VLSI) by using Stanford IC Tester, (developed at Stanford University, California). The tester has the capability of addressing, simulating, and measuring status of any pin of its test connector, to which an ICUT (IC Under Test) is attached.

The test vectors to stimulate the ICUT and reference data to analyze the response of an ICUT are extracted from ESIM files in VAX 11/780 computer system and stored on 8" floppy disks to be utilized with microcomputer. These ESIM files, typically produced during Computer Aided Design phase of a VLSI circuit, contain node data generated during its simulator run.

Continued on back page

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21 ABSTRACT SECURITY CLASSIFICATION:

22a NAME OF RESPONSIBLE INDIVIDUAL: Harold W. Carter, Lt. Col., USAF

22b TELEPHONE NUMBER *(Include Area Code)*: 513-255-6913

22c OFFICE SYMBOL: AFIT/ENG

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

The LSI-11/23 microcomputer will be used to control the functions of IC tester and provide test and reference data. The user will have the capability to gu de the course of operation by selecting various operating options in an interactive manner.

Thesis Chairman: Harold W. Carter, Lt. Col., USAF

— ORIGINATOR - SUPPLIED KEY WORDS INCLUDE:-